Oracle Service Architecture Leveraging Tuxedo

Configuration Guide

12*c* Release 1 (12.1.1)

June 2012

Oracle Service Architecture Leveraging Tuxedo (SALT) Configuration Guide, 12*c* Release 1 (12.1.1)

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 is software or related documentation that 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 END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware 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 that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software or hardware 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.

Configuring an Oracle Service Architecture Leveraging Tuxedo Application

Configuring an Oracle Service Architecture Leveraging Tuxedo Application

This section contains the following topics:

- Configuring Oracle Tuxedo Web Services
- Configuring Service Contract Discovery
- Configuring Oracle Service Architecture Leveraging Tuxedo WS-TX Support
- Oracle SALT Configuration Tool

Configuring Oracle Tuxedo Web Services

- Using Oracle Tuxedo Service Metadata Repository for Oracle Service Architecture Leveraging Tuxedo
- Configuring Native Oracle Tuxedo Services
- Configuring External Web Services
- Configuring Service Contract Discovery
- Creating the Oracle Service Architecture Leveraging Tuxedo Deployment File
- Configuring Advanced Web Service Messaging Features
- Configuring Security Features
- Compiling Oracle Service Architecture Leveraging Tuxedo Configuration

- Configuring the UBBCONFIG File for Oracle Service Architecture Leveraging Tuxedo
- Configuring Oracle Service Architecture Leveraging Tuxedo In Oracle Tuxedo MP Mode
- Migrating from Oracle Service Architecture Leveraging Tuxedo 1.1

Using Oracle Tuxedo Service Metadata Repository for Oracle Service Architecture Leveraging Tuxedo

Oracle Service Architecture Leveraging Tuxedo (SALT) leverages the Oracle Tuxedo Service Metadata Repository to define service contract information for both existing Oracle Tuxedo services and 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 `GWWS` 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 Service Architecture Leveraging Tuxedo
- Defining Service Parameters for Oracle Service Architecture Leveraging Tuxedo

Defining Service-Level Keywords for Oracle Service Architecture Leveraging Tuxedo

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 SALT and are ignored when SALT components load the Oracle Tuxedo Service Metadata Repository.

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

Service-Level Keyword	SALT Usage
`service`	The unique key value of the service. This value is referenced in the SALT WSDF file. 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. For SALT proxy services, this value typically is the Web service operation local name.
`servicemode`	Determines the service mode (i.e., native Oracle Tuxedo service or SALT proxy service. The valid values are: • `tuxedo` represents a native Oracle Tuxedo service • `webservice` represents an SALT proxy service, i.e. a service definition converted from a wsdl:operation Do not use “`webservice`” to define a native Oracle Tuxedo service. This value is always used to define services converted from external Web services.
`tuxservice`	The actual Oracle Tuxedo advertised service name. If no value is specified, then the value is the same as the value in the `service` keyword. For native Oracle Tuxedo service, SALT invokes the Oracle Tuxedo service defined using this keyword. For SALT proxy service, GWWS server advertises the service name using this keyword value.

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

Service-Level Keyword	SALT Usage
`servicetype`	Determines the service message exchange pattern for the specified Oracle Tuxedo service. The following values specify mapping rules between the Oracle Tuxedo service types and Web Service message exchange pattern (MEP): • `service` corresponds to request-response MEP • `oneway` corresponds to oneway request MEP • `queue` • corresponds to request-response MEP
`inbuf`	Specifies the input buffer (request buffer) type for the service. For native Oracle Tuxedo services, the value can be any Oracle Tuxedo typed buffer type. The following values are Oracle Tuxedo reserved buffer types: `STRING, CARRAY, XML, MBSTRING, VIEW, VIEW32, FML, FML32, X_C_TYPE, X_COMMON, X_OCTET, NULL` (input buffer is empty) **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. For SALT proxy services, the value is always `FML32`.
`outbuf`	Specifies the output buffer (response buffer with TPSUCCESS), type for the service. For native Oracle Tuxedo services, the value can be any Oracle Tuxedo typed buffer type. The following values are Oracle Tuxedo reserved buffer types: `STRING, CARRAY, XML, MBSTRING, VIEW, VIEW32, FML, FML32, X_C_TYPE, X_COMMON, X_OCTET, NULL` (input buffer is empty) **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. For SALT proxy services, the value is always `FML32`.

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

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

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

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

Defining Service Parameters for Oracle Service Architecture Leveraging Tuxedo

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.

- View32 typed buffers that support embedded VIEW32 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 SALT and are ignored when SALT components load the Oracle Tuxedo Service Metadata Repository.

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

Parameter-level Keyword	SALT Usage
`param`	Specifies the parameter name. • VIEW, VIEW32, X_C_TYPE, or X_COMMON Specifies the view structure member name in the `param` keyword. • FML, FML32 Specifies the FML/FML32 field name in the `param` keyword. • STRING, CARRAY, XML, MBSTRING, or X_OCTET SALT ignores the parameter definitions.
`type`	Specifies the data type of the parameter. **Note:** SALT does not support `dec_t` and `ptr` data types.
`subtype`	Specifies the view structure name if the parameter type is `view32`. For any other typed parameter, SALT ignores this value. **Note:** SALT requires this value if the parameter type is `view32`. This keyword is for native Oracle Tuxedo service only.

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

Parameter-level Keyword	SALT Usage
`access`	The general definition applies for this parameter. To support an Oracle Tuxedo TPFAIL scenario, the `access` attribute value has been enhanced. Original values: `in, out, inout, noaccess`. New added values: `err, inerr, outerr, inouterr`.
`count`	The general definition applies for this parameter. For SALT, the value for the `count` parameter must be greater than or equal to `requiredcount`.
`requiredcount`	The general definition applies for this parameter. The default is `1`. For SALT, the value for the `count` parameter must be greater than or equal to `requiredcount`.
`size`	This optional keyword restricts the maximum byte length of the parameter. It is only valid for the following parameter types: `STRING, CARRAY, XML, and MBSTRING` 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 SALT WSDL converter. This keyword is for 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 SALT WSDL converter according to SALT pre-defined XML-to-Tuxedo data type mapping rules. This keyword is for 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 WSDF
- Using WS-Policy Files
- Generating a WSDL File from a Native WSDF

Creating a Native WSDF

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

Each native WSDF must be defined with a unique WSDF name. A WSDF 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 It is defined in wssoapflds.h file shipped with 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 Listing 2.

Listing 2 GWWS Soap Header Translation

```
<cup:SoapHeader xmlns:cup='http://www.xxx.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 WSDF. 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 payloads and Oracle Tuxedo typed buffer conversion routines. For more information, see Using SALT Plug-ins in Oracle Service Architecture Leveraging Tuxedo *Programming Web Services* and “Configuring Plug-in Libraries” on page 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 WSDF to incorporate these separate WS-Policy files. The `location` attribute is used to specify the policy file path; both abstract and relative file path are allowed. The `use` attribute 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 `WSDF` that reference WS-Policy files:

- `<Servicegroup>`
  - If a WS-Policy file consists of Web Service Endpoint-level Assertions (for example, Reliable Messaging Assertion), the WS-Policy file applies to all endpoints serving the`<Servicegroup>` element

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

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 `WSDF` file.

Listing 6 A Sample of Defining WS-Policy Files in the WSDF 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>
      <Service name="tolower" />
    </Servicegroup>
    ....
  </WSBinding>
</Definition>
```

For more information, see “Specifying the Reliable Messaging Policy File in the WSDF File” and “Using WS-Security Policy Files”.

Generating a WSDL File from a Native WSDF

Once an Oracle Tuxedo native WSDF is created, the corresponding WSDL file can be generated using the SALT WSDL generation utility, `tmwsdlgen`. The following example command generates a WSDL file named “`app1.wsdl`” from a given `WSDF` 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 WSDF 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 Service Architecture Leveraging Tuxedo *Programming Web Services* and `tmwsdlgen` in the Oracle Service Architecture Leveraging Tuxedo *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

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 SALT installation package and is used as the default XSLT toolkit.

The 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 GoogleSearch.wsdl -o GSearch
```

Table 3 lists the Oracle Tuxedo definition files generated by SALT WSDL Converter.

Table 3 Tuxedo Definition Files generated by SALT WSDL Converter

Generated File	Description
Oracle Tuxedo Service Metadata Repository input file	SALT WSDL Converter converts each `wsdl:operation` to a Oracle Tuxedo service metadata syntax compliant service called SALT proxy service. SALT proxy services are advertised by `GWWS` servers to accept ATMI calls from Oracle Tuxedo applications.
FML32 field table definition file	Oracle Service Architecture Leveraging Tuxedo maps each `wsdl:message` to an Oracle Tuxedo FML32 typed buffer. The 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. To access an 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. **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 accordantly. For more information, see "Resolving Naming Conflict For the Generated Oracle Service Architecture Leveraging Tuxedo Proxy Service Definitions".

Table 3 Tuxedo Definition Files generated by SALT WSDL Converter

Generated File	Description
Non-native WSDF file	SALT WSDL Converter converts the WSDL file into a WSDF file, which can be deployed to GWWS servers in the SALT deployment file for outbound direction. The generated WSDF file is anon-native WSDF file. **Note:** Please do not deploy non-native WSDF files for inbound direction.
XML Schema files	WSDL embedded XML Schema and imported XML Schema (XML Schema content referenced with `<xsd:import>`) are saved locally as `.xsd` files. These files are used by GWWS servers and need to be saved under the same directory. **Note:** New XML Schema environment variables `XSDDIR` and `XSDFILES` must be set accordingly so that GWWS servers can load these `.xsd` files.

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`	SALT proxy service name. The keyword value equals to the operation local name.
	`tuxservice`	SALT proxy service advertised name in Oracle Tuxedo system. If the wsdl operation local name is less than 15 characters, the keyword value equals to the operation local name, otherwise the keyword value is the first 15 characters of the operation local name.

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`
`/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.
		Note: For more information about wsdl message and FML32 buffer mapping, see XML-to-Tuxedo Data Type Mapping for External Web Services in the SALT *Programming Web Services*.

`/wsdl:definitions`
`/wsdl:portType`
`/wsdl:operation`
`/wsdl:output`
	`outbuf=FML32`	

`/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-WSDF Element Mapping

WSDL Element	WSDF Element	Note
`/wsdl:definitions` `@targetNamespace`	`/Definition` `@wsdlNamespace`	Each wsdl:definition maps to a WSDF Definition.
`/wsdl:definitions` `/wsdl:binding`	`/Definition` `/WSBinding`	Each `wsdl:binding` object maps to a WSDF 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.

Table 5 WSDL Element-to-WSDF Element Mapping

WSDL Element	WSDF Element	Note
/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 WSDF WSBinding SOAP message style setting is equal 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 must be performed for configuring outbound Web service applications:

- Resolving Naming Conflict For the Generated Oracle Service Architecture Leveraging Tuxedo Proxy Service Definitions
- Loading the Generated SALT Proxy Service Metadata Definitions
- Setting Environment Variables for GWWS Runtime

Resolving Naming Conflict For the Generated Oracle Service Architecture Leveraging Tuxedo Proxy Service Definitions

When converting a WSDL file, unexpected naming conflicts may arise 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 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 SALT proxy service entries. Since GWWS server uses `tuxservice` values as the advertised service names, you must manually resolve the naming conflict among multiple 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 an external WSDL file into Oracle Tuxedo definitions, each `wsdl:message` is parsed and mapped as an FML32 buffer format which contains 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 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 SALT proxy service metadata definitions into the Oracle Tuxedo Service Metadata Repository through `tmloadrepos` utility. For more information, see `tmloadrepos`, in the Oracle Tuxedo Command Reference Guide.

Setting Environment Variables for GWWS Runtime

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

- Update *FLDTBLDIR32* and *FIELDTBLS32* environment variables to add the generated FML32 field table files.
- Place all excerpted XML Schema files into one directory. and set the *XSDDIR* and *XSDFILES* environment variables accordingly.
  - The *XSDDIR* and *XSDFILES* environment variables, 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 Service Architecture Leveraging Tuxedo Deployment File

The 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 SALT Deployment File Reference in the Oracle SALT Reference Guide.

Importing the WSDF Files

You should import all your required WSDF files to the 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 WSDF file must be accessible through the location specified in the SALTDEPLOY file.

Listing 7 shows how to import WSDF files in the SALTDEPLOY file.

Listing 7 Importing WSDF Files in the SALTDEPLOY File

```
<Deployment ..>
  <WSDF>
    <Import location="/home/user/simpapp_wsdf.xml" />
    <Import location="/home/user/rmapp_wsdf.xml" />
    <Import location="/home/user/google_search.wsdf" />
  </WSDF>
  ...
</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 WSDF files. Each WSBinding object is referenced using attribute "`ref=<wsdf_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="app1:app1_binding">
          <Endpoint use="simpapp_GWWS1_HTTPPort" />
          <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 can switch feature on/off or set an argument to tune server 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 on/off.	`“true”|“false”`	`“false”`
`max_backlog`	Specifies socket backlog control value.	`[1, 255]`	`20`

`max_content_length`	Specifies 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)

`thread_pool_size`	Specifies the GWWS server thread pool size.	`[1, 1024]`	`16`

Table 6 GWWS Server Level Properties

Property Name	Description	Value Range	Default

`timeout`	Specifies the network timeout in seconds.	`[1, 65535]`
		`(unit:sec)`
			`300`

`wsrm_acktime`	Specifies 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” | “RPLYRECV”`	`“NETRECV”`

Note: For more information, see “Configuring Multiple Encoding Support”.

For more information, see “Tuning the GWWS Server” in Administering 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

SALT supports multiple encoding SOAP messages and the encoding mappings between SOAP message and Oracle Tuxedo buffer. 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 "`enableMultiEncoding`" property 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 SALT Message Encoding Mapping Rules

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

Configuring System-Level Resources

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.

Note: 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. Save separate certificate PEM format files in one directory and define the directory path using the <<Certificate>/<CertPath> sub-element.

Note: The "cn" attribute of a distinguished name is used as a key for certificate lookup. Wildcards used in a name are not supported. Empty subject fields are not allowed. This limitation is also found in Oracle Tuxedo.

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 `GWWS` server is running. 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 `GWWS` server start up. The functions are registered as the implementation of the plug-in interface.

In order for the `GWWS` 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 `GWWS` 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 SALT in Oracle SALT Programming with Web Services.

Configuring Advanced Web Service Messaging Features

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

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

If you create a WS-Addressing endpoint in the `SALTDEPLOY` file or not, you can explicitly disable the Addressing capability for particular outbound services in the WSDF. 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 WSDF file. This property has no impact on any inbound services.

Listing 14 shows WSDF 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

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 WSDF. The policy file must comply with the WS-ReliableMessaging Policy Assertion Specification (February 2005).

Note: A WSDF containing a Reliable Messaging policy definition should be used by the `GWWS` 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, see the SALT WS-ReliableMessaging Policy Assertion Reference in the Oracle Service Architecture Leveraging Tuxedo *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:wsrm="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/wsrm/policy">
  <wsrm:RMAssertion>
    <wsrm:InactivityTimeout Milliseconds="600000" />
    <wsrm:AcknowledgementInterval Milliseconds="2000" />
    <wsrm:BaseRetransmissionInterval Milliseconds="500"/>
    <wsrm:ExponentialBackoff />
    <beapolicy:Expires Expires="P1D" />
    <beapolicy:QOS QOS=”ExactlyOnce InOrder" />
  </wsrm:RMAssertion>
</wsp:Policy>
```

Specifying the Reliable Messaging Policy File in the WSDF File

You must reference the WS-ReliableMessaging policy file at the `<Servicegroup>` level in the native WSDF file. Listing 16 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” />
      <Service ... />
      <Service ... />
      ...
    </Servicegroup ...>
  </WSBinding>
</Definition>
```

Note: Reliable Messaging in SALT does not support process/system failure scenarios, which means SALT does not store the message in a persistent storage area. 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

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

- Configuring Transport-Level Security
- Configuring Message-Level Web Service Security
- Configuring SAML Single Sign-On

Configuring Transport-Level Security

SALT provides point-to-point security using SSL link-level security and supports HTTP basic authentication mechanisms 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, see "Configuring Certificates".

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

Configuring Inbound HTTP Basic Authentication

SALT depends on the Oracle Tuxedo security framework for Web Service client authentication. There is no special SALT configuration required to enable inbound HTTP Basic Authentication.

If the Oracle Tuxedo system requires user credentials, HTTP Basic Authentication is an alternative for Web Service client programs to carry user credentials.

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 QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

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. SALT does not make group information available for Web service clients.

Configuring Outbound HTTP Basic Authentication

SALT supports authentication plug-in development to prepare user credentials for outbound HTTP Basic Authentication. Outbound HTTP Basic Authentication is configured at Endpoint-level. If an outbound Endpoint requires a user profile in the HTTP message, you must specify the HTTP Realm for the HTTP endpoint in the WSDF file. The `GWWS` server invokes the authentication plug-in library to prepare usernames and passwords, and sends 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 sent to an outbound endpoint using <`Realm`> element, the `GWWS` 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”.and Programming Outbound Authentication Plug-ins in the Oracle Service Architecture Leveraging Tuxedo *Programming Web Services*.

Configuring Message-Level Web Service Security

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

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

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 SALT.

Table 8 lists the default WS-Security Policy files bundled by SALT.

Table 8 WS-Security Policy Files Provided By 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
`wssp1.2-Wss1.1-X509V3-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 (with the exception of the WS-Security Policy 1.2 `UserToken` file), can be referenced using`<Servicegroup>` or `<Service>` elements in the native `WSDF` file. The WSSP 1.2 `UserToken` file can only be referenced using`<Servicegroup>`.

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 requires the SOAP:Body part of message to be signed with the X.509 token. The sample “wsseapp” shows how to clip the WSSP 1.2 UserToken file used in the <Service> element.

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 using the “location” attribute of the <Policy> element. A prefix “salt:” means an 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 the <Servicegroup> level, it applies 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 the SOAP:Body of the output message is not signed.

Configuring SAML Single Sign-On

- Transport Protection

- SAML Key File

SALT supports SAML 1.1 and SAML 2.0 Single Sign-On (SSO). You can use Single Sign-On to process a secure incoming request by performing authentication on behalf of the end user, without having to request their credentials.

The SALT implementation of SAML SSO supports the sender-vouches confirmation method. With this method, SALT represents a back-end system, and a Web Service intermediary sits between the back-end and the end user. In this case, the Web Service intermediary "vouches" for the end user using SAML token mechanisms.

Transport Protection

Although it is not required to use TLS/SSL as a transport to carry an SAML security token to access Oracle Tuxedo through GWWS, it is recommended that the Web Service intermediary use TLS/SSL to access Oracle Tuxedo through GWWS using an SAML security token. The use of TLS/SSL ensures the SOAP message content from being disclosed or modified without detection This is particularly important when accessing Oracle Tuxedo services through a wide area network outside of a fire wall.

SAML Key File

The public key certificate of trusted SAML assertion issuers must be located in the $APPDIR directory. These certificates must be in PEM format. The name of the certificate must reflect the issuer name. For instance, if the issuer id is `"ws_1"` then the certificate name should be `ws_1.pem`.

However, for long issuer names the key file provides the ability to correlate between the real issue name and its local reference name so that the `PEM` file name can be much more concise but still remain useful to the administrator.

For example, if the assertion issuer name is `web.abc.com/saml/authenticator`, then the `PEM` file name for its public key certificate can be called `"abc.pem"` instead of `"www.abc.com/saml/authenticator.pem"`.

This is especially useful when in a UNIX environment where the "/" symbol also works as a path separator. This translation is required when confusion like this may arise.

The key file name is fixed to `"saml_key.meta"`. It should be located in the same file folder specified by `"CertPath"`. This file should be protected by the file system and is in XML format.

This section contains the following topics:

- Key File Format

- File Information
- GWWS Key
- Assertion Issuer Information
- Key File Generation
- Procedure to Manage Key File
- WS-Policy Files
- Mapping SAML Elements with Oracle Tuxedo Security

Key File Format

The key file is an XML file. There are three types of information stored in this file:

file information, GWWS key, and issuer information.

Note: You should not modify this file manually since this will cause the file to fail integrity checking.

File Information

The file information section contains the version number of the tool generated this file, a random key, administrative password, and digital signature.

GWWS Key

This GWWS key section contains one GWWS symmetric key. There can be only one symmetric configured for GWWS to simplify the validation task. This key is encrypted with obfuscated key. This section is optional and is missing if no GWWS symmetric key is configured.

In MP configuration with multiple GWWS on different machine nodes, this file needs to be replicated on each node; however, if a different GWWS key is desired, then a similar key file but with a different GWWS key record can be copied to a different node.

Assertion Issuer Information

This section contains multiple records, one for each trusted assertion issuer. It contains issuer identifier, local issuer identifier, symmetric key, and whether a public key certificate also exists or not.

The issuer identifier is the value presented in the `"issuer"` attribute of `"<saml:Assertion>"` element in the WSSE security header.

The local issuer identifier is the abbreviated name for the issuer. The purpose is to make any long issuer identifier become shorter and easier to memorize, but still remain locally unique. This data is optional; if it exists and a certificate also exists, then the certificate must take the name of this local issuer identifier with 'pem" as file extension.

The symmetric key is the shared secret that issuer used to sign the assertion. This data is optional. The length of the key also dictates which algorithm can be used for signing.

The public key certificate exists field tells whether a public key certificate exists. If it exists, the certificate should be located in the folder specified by the "CertPath" element. This field can be true while the symmetric key field also exists. At runtime, GWWS detects which key to use to validate the signature.

Key File Generation

A new command is added to wsadmin to manage the key file. This new command is used to generate new key file, add new record, delete existing record, and modify record. The name of the file it managed is "saml_key.meta" in the current working directory.

To create the key file issues the following wsadmin command:

```
saml create -p password
```

Where the "-p password" is for the administrative password to access the newly created key file. A key file with name "saml_key.meta" is created in the current working directory.

To add a trusted issuer, input the following command:

```
saml add -i -n authority.abc.com -l abc -c -p password
```

Where "-i" tells it to add an issuer with name "authority.abc.com" with short local reference name "abc" and the access password to access the key file. The key file saml_key.meta" must exist in current working directory. Since "-c" option is given, a public key certificate named "abc.pem" must exist in the "CertPath".

For more information, see wsadmin the SALT Command Reference.

Procedure to Manage Key File

The following procedure describes a SALT administrator setting up GWWS to be able to handle SAML assertion for the first time.

1. Change directory to $APPDIR and start wsadmin.
2. Use "saml create" command to create the key file.
3. Use "saml add -g" command to add GWWS record.
4. Use "saml add -i" command to add trusted assertion issuer record for every trusted assertion issuer.

5. Copy the file "`saml_key.meta`" to the directory described in the SALT deployment descriptor file "`CertPath`" element under "Certificate".
6. Change directory to Oracle Tuxedo application domain, and use "`tmboot -y`" to boot the Oracle Tuxedo application domain.

In MP mode configuration, it is possible to have a different GWWS record in the key file for a different GWWS instance. The following procedure creates the key file for a GWWS instance on a different node.

1. Copy the original key file to different directory or machine.
2. Use "`saml delete -g`" to delete existing GWWS record.
3. Use "`saml add -g`" to add a different GWWS record.
4. Boot Oracle Tuxedo.

WS-Policy Files

SALT includes a number of WS-Policy files that you can use for configuring services for SAML SSO as listed in Table 9

Table 9 SAML SSO Policy Files

File Name	Purpose
Wssp1.2-2007-Saml1.1-SenderVouches.xml	SAML 1.1 support
Wssp1.2-2007-Saml2.0-SenderVouches.xml	SAML 2.0 support

The above files can be referenced at the `<ServiceGroup>` or `<Service>` level in the native WSDF file.

This policy may be combined with other WS-Security policies (such as inbound body signature). For more information, see Configuring Message-Level Web Service Security.

For example, Listing 19 shows the SAML 1.1 policy file with supported capabilities outlined.

Listing 19 SAML 1.1 Policy File

```
<?xml version="1.0"?>

<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">

     <sp:AsymmetricBinding>
        <wsp:Policy>
          <sp:InitiatorToken>
            <wsp:Policy>
              <sp:X509Tokensp:IncludeToken="http://docs.oasis-open.org/ws-
sx/ws-securitypolicy/200512/IncludeToken/Always">
            <wsp:Policy>
              <sp:WssX509V3Token10/>
               </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:InitiatorToken>
      <sp:RecipientToken>
        <wsp:Policy>
          <sp:X509Token
sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512
/IncludeToken/Never">
           <wsp:Policy>
             <sp:WssX509V3Token10/>
           </wsp:Policy>
         </sp:X509Token>
       </wsp:Policy>
      </sp:RecipientToken>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:Basic256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:Layout>
        <wsp:Policy>
          <sp:Lax/>
        </wsp:Policy>
      </sp:Layout>
      <sp:IncludeTimestamp/>
      <sp:ProtectTokens/>
    </wsp:Policy>
  </sp:AsymmetricBinding>
  <sp:SignedSupportingTokens>
    <wsp:Policy>
      <sp:SamlToken
sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702
/IncludeToken/AlwaysToRecipient">
        <wsp:Policy>
          <sp:WssSamlV20Token11/>
        </wsp:Policy>
      </sp:SamlToken>
    </wsp:Policy>
  </sp:SignedSupportingTokens>
</wsp:Policy>
```

Mapping SAML Elements with Oracle Tuxedo Security

Table 10 lists what optional SAML assertion elements must present.

Table 10 Optional SAML Assertion Elements

Oracle Tuxedo Security and SAML Assertion Correspondence		
Oracle Tuxedo SECURITY Level	**Additional SAML Assertion Elements Required**	**Access Principal**
NONE	None	Anonymous, Subject/NameID
APP_PW	None	Anonymous, Subject/NameID
USER_PW	Subject	Subject/NameID
ACL	Subject	Subject/NameID
MANDATORY_ACL	Subject	Subject/NameID

In NONE and APP_PW cases, if the optional element "Subject" exists, then "NameID" is used to access Oracle Tuxedo. If the optional element "Subject" does not exist, then the client assumes anonymous user identity to access Oracle Tuxedo. If the anonymous access is not allowed (i.e. no credential mapping for anonymous), then the request fails.

If the SAML assertion does not contain a "Subject" element and Tuxedo SECURITY level is configured at USER_PW, ACL, or MANDATORY_ACL, then the request is rejected.

Compiling Oracle Service Architecture Leveraging Tuxedo 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 WSDF 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 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, see wsloadcf reference in the Oracle Service Architecture Leveraging Tuxedo *Reference Guide*.

Configuring the UBBCONFIG File for Oracle Service Architecture Leveraging Tuxedo

After configuring and compiling the SALT configuration, the Oracle Tuxedo UBBCONFIG file needs to be updated to apply SALT components in the Oracle Tuxedo application. Table 11 lists the UBBCONFIG file configuration tasks for SALT.

Table 11 UBBCONFIG File Configuration Tasks for 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

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 20 lists a segment of the UBBCONFIG file that shows how to define TMMETADATA servers in an Oracle Tuxedo application.

Listing 20 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 entire 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 Oracle Tuxedo 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 21 lists a segment of the UBBCONFIG file that shows how to define GWWS servers in an Oracle Tuxedo application.

Listing 21 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.

SALT configuration information is pre-compiled with wsloadcf to generate theSALTCONFIG file binary. GWWS server reads the SALTCONFIG file at start up.The

SALTCONFIG environment variable must be set correctly with the SALTCONFIG file entity before booting GWWS servers.

Option "-c" is deprecated in the current version 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 must plan and update Oracle Tuxedo system limitations defined in the UBBCONFIG file according to your SALT application requirements.

Tip: Define an adequate MAXSERVERS number in the *RESOURCES section

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: Define an adequate 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: Define an adequate MAXACCESSERS number in the *RESOURCES section

The 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: Define an adequate MAXWSCLIENTS number in the *MACHINES section

When the GWWS server operating in the inbound direction, each Web Service client is deemed a workstation client in the Oracle Tuxedo system; therefore, MAXWSCLIENTS must be configured with a valid number in the UBBCONFIG file for the machine where the GWWS server is deployed. The number is shared.

Configuring Certificate Password Phrase For the GWWS Servers

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

When GWWS servers are specified with certificate-related features, they are required to read the private key file and decrypt it using the password phrase. To configure a password phrase for each GWWS server, the keywords *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 tmloadcf, 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 SALT deployment file. All the GWWS servers defined in the SALT deployment file must be provided the same password phrase for the private key file decryption.

Listing 22 shows a segment of the UBBCONFIG file that defines a security password phrase for the GWWS servers.

Listing 22 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 12cR1 documentation.

Configuring Oracle Tuxedo Authentication for Web Service Clients

SALT GWWS servers rely on Oracle Tuxedo authentication framework to check the validity of the Web Service clients. If your existing Oracle Tuxedo application is already applied, Web Service clients must send user credentials using one of the following:

- 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 SALT, you must configure Oracle Tuxedo authentication in the Oracle Tuxedo domain.

For more information, see Administering Authentication in the *Oracle Tuxedo 12cR1 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 must configure the Oracle Tuxedo Security level as USER_AUTH, ACL or MANDATORY_ACL in the UBBCONFIG file.

Listing 23 shows a segment of the UBBCONFIG file that defines security-level ACL in the UBBCONFIG file.

Listing 23 Security-Level ACL Defined in the UBBCONFIG File For Outbound HTTP Basic Authentication

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

Configuring Oracle Service Architecture Leveraging Tuxedo In Oracle Tuxedo MP Mode

To set up GWWS servers running on multiple machines within an 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 GWWS 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 Service Architecture Leveraging Tuxedo 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 GWWS servers with SALT 1.1 Configuration File
- Adopting SALT 2.0 Configuration Style by Converting SALT 1.1 Configuration File

Running GWWS 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. This is supported in SALT 2.0.

The SALT configuration compiler utility, `wsloadcf`, supports loading 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 must perform the following steps:

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

Note: If you have more than one SALT 1.1 configuration files defined in an Oracle Tuxedo domain, you must follow steps 1 - 3 to generate more binary `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 `SALTCONFIG` from a SALT 1.1 configuration file, it also converts this SALT 1.1 configuration file into one `WSDF` file and one `SALTDEPLOY` file.

It is highly recommended to start using the SALT 2.0 styled configuration once you get the converted files from SALT 1.1 configuration. If you want to incorporate more than one SALT 1.1 configuration file into one SALT 2.0 deployment, you must manually edit the `SATLDEPLOY` file for importing the other WSDF files.

Listing 24 shows the converted `SALTDEPLOY` file and `WSDF` file from a given SALT 1.1 configuration file.

Listing 24 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 WSDF file and deployment file are shown in Listing 25 and Listing 26 respectively.

Listing 25 Converted WSDF File for SALT 1.1 Configuration File (simpapp.xml.wsdf)

```
<Definition name="simpapp" wsdlNamespace="urn:simpapp.wsdl"
  xmlns=" http://www.bea.com/Tuxedo/WSDF/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 26 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 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 can be stored in the metadata repository, or output to a text file (which is then loaded to the metadata repository using `tmloadrepos)`. SALT uses the `tmscd` command to control the service contract runtime collection. For more information, see `tmscd` in the Oracle Service Architecture Leveraging Tuxedo *Command Reference Guide*.

Generated service contract information contains the 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 the 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. SALT can generate a detailed description by view name when using metadata repository.

Note: Patterns flagged with `autodiscovery` (see Table 12) are compared.

If the same `autodiscovery` pattern already exists in the metatdata 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 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 is the same as 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 the request is 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 the metadata repository. The file contains a service header section and a parameter section for each parameter. Service header contains items listed in Table 12. 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 13.

Table 12 Service Level Attributes

Keyword (abbreviation)	Sample Value	Description
service(sv)	TOUPPER_1	<RealServiceName>_<SequenceNo>.
tuxservice(tsv)	TOUPPER	The service name.
servicetype(st)	service|oneway	one way if TPNOREPLY is set.
inbuf(bt)	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.
outbuf(BT)	FML32	set to "NULL" if it is an error reply.
errbuf(ebt)	STRING	present only when it is an error reply.

Table 12 Service Level Attributes

Keyword (abbreviation)	Sample Value	Description
`inview`		View name. Present only when inbuf is of type VEW or VIEW32.
`outview`		View name. Present only when outbuf is of type VIEW or VIEW32.
`errview`		View name. Present only when errbuf is of type VIEW or VIEW32.
`autodiscovery`	`true`	Set to "true".

Table 13 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 Service Architecture Leveraging Tuxedo 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 WSDF additions pseudo-schema Appendix.

For additional information, see the 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 27.

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 27 TLOG Element Added to SALTDEPLOY File

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

Registration Protocol

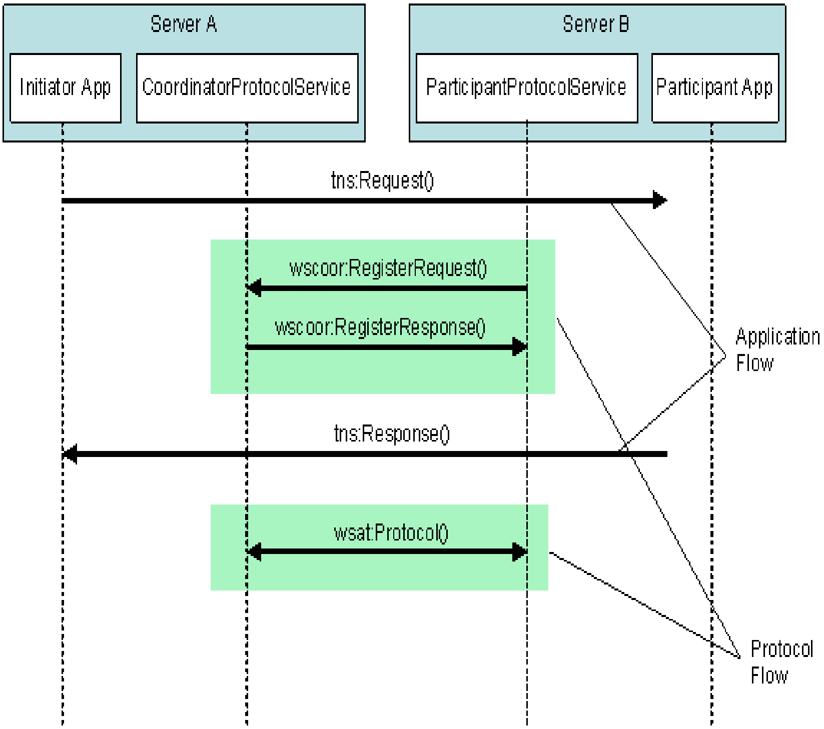
Oracle Tuxedo-based services are registered with a Durable 2PC protocol with coordinators.

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 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 g 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 `TPETRAN` 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 g 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 WSDL files, 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 `SALTDEPLOY` source.

Inbound Transactions

For inbound transactions, no particular behavior change takes place at runtime. The client consuming the WSDL takes the decision based on the configured value and runtime behavior is the same for the normal cases.

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.

Oracle Service Architecture Leveraging Tuxedo Configuration Tool

The SALT configuration tool allows you to view and modify your configuration. Oracle Tuxedo services can be exposed as Web Services without having to edit configuration files.

Enabling the Oracle Service Architecture Leveraging Tuxedo Configuration Tool

- GWWS Option

- Security

Note: With Internet Explorer 9, web pages are displayed without CSS information by default. In order to display the pages properly, Internet Explorer 8 document compatibility mode should be used. Do the following:

- Click on tools (little gear icon on the right) then select "Developer Tools" or hit F12
- Select "Document mode" and "IE8 standards"

GWWS Option

Web administration is disabled by default. In order to enable admin capabilities, the GWWS server must be configured as such in `UBBCONFIG`, using the `-a` option as follows:

```
-a <scheme>://<host>:<port>
```

`<scheme>` can be 'http' or 'https'. If using 'https', GWWS must be configured for SSL capabilities by adding private-key and certificates in the same manner as securing application Web Services with SSL.

`<host>` is the name or IP address of the admin URL listening endpoint.

`<port>` is the port value of the admin URL listening endpoint.

Use the following URL to access the Web Admin Console:

```
<scheme>://<host>:<port>/admin
```

For example: `https://server.company.com:3333/admin`.

Security

Note: It is recommended that you use SSL/TLS to protect user name and password in order to integrate the SALT Configuration Tool with Oracle Tuxedo security.

For Oracle Tuxedo application domains that requires `ACL` or `MANDATORY_ACL` security, a console service must be configured in the Oracle Tuxedo security data files. This added information is used for Oracle Tuxedo access control to the Configuration Tool service. By default, the Configuration Tool service name is `"SALTWEBCONSOLE"`, but you can modify it using the GWWS option `"-C <CONSOLE SERVICE NAME>"`. For example:

```
GWWS    SRVGRP=GROUP1 SRVID=3
                CLOPT="-A -- -iGWWS1 -a
http://server.company.com:3333/admin -C CONSOLE"
```

This tells GWWS to use `"CONSOLE"` as the service name for access control.

Note: You should also use "tpacladd" to add this Web Console service into the security data file.For example: `$ tpacladd -g 1000 CONSOLE`.

This will add `CONSOLE` as an Oracle Tuxedo `SERVICE` into the security data file and restrict the access only to user belongs to the group with group id 1000.

Configuring Configuration Tool Security

No Security

Without configuring `SECURITY` in the `"*RESOURCES"` section of the `UBBCONFIG` file or configuring it with a value of `"NONE"`, no security is used for accessing the SALT Configuration Tool. Anyone who knows the URL of the tool can access it. Listing 3 shows a `UBBCONFIG` file `"*RESOURCES"` section example.

Listing 3 No Security UBBCONFIG *RESOURCES Section Example

```
*RESOURCES
IPCKEY                  15301
DOMAIN                  mydomain
MASTER                  machine1
MAXACCESSERS            50
MAXSERVERS              10
MAXSERVICES             40
MODEL                   SHM
LDBAL                   N
```

Application Password Security

Configuring `SECURITY` in the `"*RESOURCES"` section with a value of `APP_PW` causes Oracle Tuxedo application password security to be enabled. Users who want to access the SALT configuration tool are requested to present this password; failure to do so results in denied access. Listing 4 shows a `UBBCONFIG` file `"*RESOURCES"` section example.

Listing 4 Application Password Security UBBCONFIG *RESOURCES Section Example

```
*RESOURCES
IPCKEY                  15301
DOMAIN                  mydomain
MASTER                  machine1
MAXACCESSERS            50
MAXSERVERS              10
MAXSERVICES             40
MODEL                   SHM
LDBAL                   N
SECURITY                APP_PW
```

User Authentication Security

Configuring `SECURITY` in the `"*RESOURCES"` section with a value of `USER_AUTH` causes Oracle Tuxedo user authentication security to be enabled. To access the SALT configuration tool users are requested to present a valid Oracle Tuxedo user name and password; failure to do so results in denied access. Listing 5 shows a `UBBCONFIG` file `"*RESOURCES"` section example.

Listing 5 User Authentication Security UBBCONFIG *RESOURCES Section Example

```
*RESOURCES
IPCKEY                  15301
DOMAIN                  mydomain
MASTER                  machine1
MAXACCESSERS            50
MAXSERVERS              10
MAXSERVICES             40
MODEL                   SHM
```

```
LDBAL                   N
SECURITY                USER_AUTH
```

A user can be added using the "`tpusradd`" command. The following example adds user "`tom`" to the group with group id 1000 in the Oracle Tuxedo application domain.

```
$ tpusradd -u 2503 -g 1000 tom
```

Access Control List Security

Configuring `SECURITY` in the "`*RESOURCES`" section with a value of `ACL` causes Oracle Tuxedo access control list security to be enabled. Anyone who wants to access the SALT configuration tool is requested to present a valid Oracle Tuxedo user name and password that belongs to the group(s) allowed to access the Web Console; failure to do so results in denied access. Listing 6 shows a `UBBCONFIG` file "`*RESOURCES`" section example.

Listing 6 Access Control List Security UBBCONFIG *RESOURCES Section Example

```
*RESOURCES
IPCKEY                  15301
DOMAIN                  mydomain
MASTER                  machine1
MAXACCESSERS            50
MAXSERVERS              10
MAXSERVICES             40
MODEL                   SHM
LDBAL                   N
SECURITY                ACL
```

Access control to the configuration tool can be added using the "`tpacladd`" command. The following example adds Configuration Tool service "`SALTWEBCONSOLE`" to the access control list in an Oracle Tuxedo application domain.

```
$ tpacladd -g 1000 SALTWEBCONSOLE
```

If the service is not added to the Oracle Tuxedo access control security data file, any user with a valid Oracle Tuxedo user name and password can access the SALT Web Console.

Mandatory Access Control List Security

Configuring SECURITY in the "*RESOURCES" section with a value of MANDATORY_ACL causes Oracle Tuxedo access control list security to be enabled. Anyone who wants to access the SALT configuration tool is requested to present a valid Oracle Tuxedo user name and password that belongs to the group(s) allowed to access the configuration tool; failure to do so results in denied access. Listing 7 shows a UBBCONFIG file "*RESOURCES" section example.

Listing 7 Mandatory Access Control List Security UBBCONFIG *RESOURCES Section Example

```
*RESOURCES
IPCKEY                  15301
DOMAIN                  mydomain
MASTER                  machine1
MAXACCESSERS            50
MAXSERVERS              10
MAXSERVICES             40
MODEL                   SHM
LDBAL                   N
SECURITY                MANDATORY_ACL
```

Access control to the configuration tool can be added using the "tpacladd" command. The following example adds the configuration tool service "SALTWEBCONSOLE" to the access control list in the Oracle Tuxedo application domain.

```
$ tpacladd -g 1000 SALTWEBCONSOLE
```

If the service is not added to the Oracle Tuxedo access control security data file, then no one can access the SALT Web Console.

See Also

- `tmadmin`
- `tmloadrepos`
- `ubbconfig`
- WSDF documentation
- SALT Programming Guide
- SALT Reference Guide
- SALT Interoperability Guide