Oracle® Tuxedo JCA Adapter

Users Guide

12*c* Release 1 (12.1.1)

August 2012

ORACLE®

Contents

Oracle Tuxedo JCA Adapter Users Guide

Oracle Tuxedo JCA Adapter Users Guide

This chapter contains the following topics:

- Overview

- Prerequisites

- Installing the Oracle Tuxedo JCA Adapter

- Using the Oracle Tuxedo JCA Adapter

- Oracle Tuxedo JCA Adapter Deployment

- Oracle Tuxedo JCA Adapter Management

- Oracle Tuxedo JCA Adapter Fail Over Capability

- Oracle Tuxedo JCA Adapter Security

- JATMI Outbound Conversation

- Using the JDeveloper SOA Suite

Overview

The Oracle Tuxedo JCA Adapter (Tuxedo JCA Adapter) is based on JCA Connector Architecture. It connects a Java application server to a GWTDOMAIN gateway in an Oracle Tuxedo application domain to provide bidirectional access to/from Oracle Tuxedo. The connection between the Tuxedo JCA Adapter and a GWTDOMAIN gateway can be a simple or

proprietary (but high performance) TCP socket connection, Link-Level Encryption, or an industry standard SSL/TLS. Over connection types, the Tuxedo JCA Adapter creates an application-level TDOMAIN session and runs the TDOMAIN Protocol to communicate with an Oracle Tuxedo GWTDOMAIN gateway.

The TDOMAIN protocol is a robust and powerful protocol; it supports synchronous/asynchronous request replies, conversations, Oracle Tuxedo queue operations, identity propagation, transaction propagation, and domain authentication.

The Tuxedo JCA Adapter supports both Link-Level and Service-Level Failover. Link-Level Failover is designed for a multi-hosted server that, when the primary network interface is not working, can switch to use a different network interface on the same machine. Link-Level Failover can also be used on two Oracle Tuxedo application domains that offer the same set of services through the GWTDOMAIN gateway. When a primary domain is shutdown or unavailable, the backup domain is used to create a connection and TDOMAIN session.

Service-Level Failover provides the ability to forward the service request to a backup remote access point when the primary access point is not available. Link-Level Failover and Service-Level Failover act on different objects, connections vs. services. They are independent of each other; however, they can work together to provide high availability.

Access Point

There are two access point types:

- Local Access Point, and

- Remote Access Point.

Both access point types can have a `name` and `identifier`. The `name` is used to identify an access point configuration entry in a configuration; it must be unique to the configuration. The `identifier` is used to specify an access point during TDOMAIN Session establishment; it must be globally unique within all interconnected domains.

Both access point types can also have one or more network addresses associated with it. The only exception is the default Local Access Point which is not configured, but instead, is dynamically generated and registered with a Remote Access Point that is capable of accepting dynamic registration.

The Local Access Point corresponds to `DM_LOCAL_DOMAINS`, and Remote Access Point corresponds to `DM_REMOTE_DOMAINS` in the Oracle Tuxedo Domain `BDMCONFIG` configuration file.

Session and Session Profile

The Tuxedo JCA Adapter creates an application session with a GWTDOMAIN gateway called TDOMAIN Session. It involves one Local Access Point and one Remote Access Point. A TDOMAIN Session sits on top of a connection and handles protocol exchanges between two entities that speak TDOMAIN Protocol.

Note: A session must be configured for a Local Access Point to enable it to communicate with a Remote Access Point.

A Tuxedo JCA Adapter Session Profile describes session characteristics. A Session Profile contains configurable information (such as Connection Policy, Security, BlockTime, etc.).A session must have a session profile associated with it; if not configured, the default session profile is used.

Import and Export

An Oracle Tuxedo resource must be imported to be accessed by a Java client using either CCI or JATMI. The imported resource has a `name` and a `RemoteName`. The "`name`" is what a JCA client uses to tell the Tuxedo JCA Adapter which remote resource configuration entry to use and the "`RemoteName`" is the actual name the Remote Access Point uses to reference its exported resource. When more than one TDOMAIN session provides the same service using the same configuration entry, the Tuxedo JCA Adapter performs load balancing among all imported Oracle Tuxedo resources with the same name.

There are two load balancing algorithms supported by the Tuxedo JCA Adapter; by default the load balancing algorithm is `RoundRobin`, but it can be changed to `Random`. If the Tuxedo JCA Adapter is not configured for import, then all client requests are distributed among all the Remote Access Points configured using load balance; this is called Default Import.

A Java resource must be exported to be accessed by a c1lient. Three types of resources can be exported:

- POJO,

- EJB, and

- connector-based MDB.

Once these resources are configured and deployed, an Oracle Tuxedo ATMI client can access anyone of them from a partner Oracle Tuxedo application domain.

Configuration Styles

The Tuxedo JCA Adapter requires user configuration (this configuration is in addition to the Resource Adapter Deployment Descriptor). At the very least, a Remote Access Point must be configured. There are three different ways to configure the Tuxedo JCA Adapter:

- Resource Adapter Deployment Descriptor-Only Based Configuration
- Connection Factory Based Configuration
- Configuration File Based Configuration

Resource Adapter Deployment Descriptor-Only Based Configuration

Resource Adapter Deployment Descriptor-only based configuration is aimed for simple and easy configuration. All the configuration information is in the Resource Adapter Deployment Descriptor (also known as `ra.xml`). The configuration includes Local Access Point, Remote Access Point, and Session Profile. The sessions are implicitly defined from every Local Access Point to every Remote Access Point. They all share one session profile.

Connection Factory Based Configuration

Connection Factory based configuration is middle ground in terms of complexity and ability. It allows you to configure extra information about a connection factory. All connections created using that factory inherit the same configuration information. The configuration information for each factory is independent of each other. The effective configuration information for each factory is the combination of the configuration information in the Resource Adapter Deployment Descriptor and the Connection Factory Configuration (Connection Factory configuration takes precedence). In some Java Application Servers, these configuration attributes can be configured using the console.

Configuration File Based Configuration

A Configuration File based configuration requires the creation of a separate XML based configuration file. This configuration method is the most complete way to configure the Tuxedo JCA Adapter. If any information resides in the configuration that requires extra privacy and protection, the Tuxedo JCA Adapter provides a tool (`com.oracle.tuxedo.tools.DMConfigChecker`), that must be used to encrypt those elements and generates a properly obfuscated key file used at runtime to decode those elements. This tool is provided by Tuxedo JCA Adapter.

Note: Mixing two or more configuration styles is not supported, you can only choose one of these styles.

Prerequisites

Installing and deploying the Tuxedo JCA Adapter requires the following prerequisites:

- Access to JDK 1.5 or later

- 5 MB free disk space

- A text editor or XML editor

Installing the Oracle Tuxedo JCA Adapter

The Tuxedo JCA Adapter release 12c is distributed in one.ZIP file (downloaded from the Oracle Web site). This `.ZIP` file contains four files which include a complete Tuxedo JCA Adapter and an Tuxedo JCA Adapter SOA configuration wizard for JDeveloper.

Copy the downloaded `.zip` file to a target directory with correct read and write access control. Unzip the file and browse the contents.

Table 1 lists the Tuxedo JCA Adapter resource `.zip` file contents.

Table 1 Tuxedo JCA Adapter Resource Archive File Content

Resource Name	Description
`com.oracle.tuxedo.TuxedoAdapter.rar`	The complete Tuxedo JCA Adapter resource archive.
`soa-config.xml`	SOA configuration file with updated Tuxedo JCA Adapter configuration wizard.
`tuxedoAdapter-config.xml`	Tuxedo JCA Adapter configuration.
`com.oracle.tuxedo.adapter.tja_soa_1.3.0.0.jar`	The Tuxedo JCA Adapter configuration wizard for SOA.

You can copy the resource archive (RAR) file to packaging directory with correct read and write permission if desired. Un-jar the resource archive to browse the contents and view the standard deployment descriptor.

Table 2 lists the Tuxedo JCA Adapter resource archive file content.

Table 2 Tuxedo JCA Adapter Resource Archive File Content

Resource Name	Description
`com.oracle.tuxedo.a dapter_1.4.0.0.jar`	JAR file containing the Tuxedo JCA Adapter classes.
`com.bea.core.jatmi_ 2.0.1.0.jar`	JAR file containing Java ATMI buffer types and interfaces. **Note:** For WebLogic Server, this jar file must be exported using `EXT_PRE_CLASSPATH` to replace the one included with WebLogic Server installation. This environmental variable must be set before starting WebLogic Server.
`com.bea.core.i18n_1 .4.0.0.jar`	The I18N utility classes **Note:** WebLogic Server users should not set this jar file in the `CLASSPATH` if the WebLogic Server version is 11gR1 or above.
`javax.transaction_1 .0.0.0_1-1.jar`	JTA 1.1 classes.
`javax.ejb_3.0.1.jar`	EJB 3.0 support classes.
`adapter.properties`	The Tuxedo JCA Adapter message catalogue.
`adapter_ja.properti es`	The Tuxedo JCA Adapter Japanese Message catalogue.
`tja.xsd`	The Tuxedo JCA Adapter configuration XML schema.
`dmconfig.xml`	An example Tuxedo JCA Adapter configuration file.
`META-INF/ra.xml`	An example Resource Adapter Deployment Descriptor that uses Factory-based configuration style
`META-INF/client-sid e.ra.xml`	An example Resource Adapter Deployment Descriptor that uses client-side only configuration style

Table 2 Tuxedo JCA Adapter Resource Archive File Content

Resource Name	Description
`META-INF/server.ra.xml`	An example Resource Adapter Deployment Descriptor that uses the `dmconfig` file as its configuration.
	If you choose to use `dmconfig` to configure the Tuxedo JCA Adapter, you can rename this sample deployment descriptor to `ra.xml` and modify it to point to your dmconfig file. You can use the sample `dmconfig.xml` as base for your `dmconfig` file.
`META-INF/weblogic-ra.xml`	An example WebLogic Server deployment descriptor for resource adapter. It defines the connection factory interface and `JNDI` name for the factory.
`META-INF/sample.weblogic-ra.xml`	The sample WebLogic Server deployment descriptor for Tuxedo JCA Adapter that contains factory-based configuration
`META-INF/MANIFEST.MF`	The Tuxedo JCA Adapter `Manifest` file

Note: The `com.bea.core.i18n_1.4.0.0.jar` is for Java application servers other than WebLogic Server; do not set it to replace WebLogic Server. The only file that needs to be overridden is the WebLogic Server `com.bea.core.jatmi_xxxx.jar` file.

The Tuxedo JCA Adapter Resource Archive contains the most recent fix to enable the Tuxedo JCA Adapter. For all other Java Application Servers, no `JATMI` file needs to be overridden.

Application Server Support

The Tuxedo JCA Adapter release 12c supports the following platforms:

- WebLogic Server version 11gR1 & 12c
- WebSphere 7.0 & 8.0.
- Oracle Tuxedo 11gR1 and later

Installing the Oracle Tuxedo JCA Adapter SOA Configuration Wizard for JDeveloper

There are three files zipped together with the Tuxedo JCA Adapter for the SOA configuration wizard for JDeveloper. This feature allows you to configure an SOA application that can access

services residing inside an Oracle Tuxedo Application domain. To do this, these three files must be properly installed so that JDeveloper recognizes the Tuxedo JCA Adapter configuration wizard.

To install the Tuxedo JCA Adapter SOA configuration wizard patch, do the following steps:

1. Copy `com.oracle.tuxedo.adapter.tja_soa_1.3.0.0.jar` to `$JDEVELOPER_HOME/jdev/lib/patches`

2. Copy `tuxedoAdpater-config.xml` to `$JDEVELOPER_HOME/integration/seed/soa/configuration`

3. Save `$JDEVELOPER_HOME/integration/seed/soa/configuration/soa-config.xml` if needed.

4. Copy `soa-config.xml` from the Tuxedo JCA Adapter .zip file to `$JDEVELOPER_HOME/integration/seed/soa/configuration`

JDeveloper is installed in the `$JDEVELOPER_HOME` directory.

JDeveloper Support

The Tuxedo JCA Adapter release 12c contains a configuration wizard for SOA. It supports JDeveloper 11gR1 Release1 (11.1.1.4.0).

Using the Oracle Tuxedo JCA Adapter

The Oracle Tuxedo Adapter can be configured one of three ways:

- Using an XML-based configuration text file (also called dmconfig).

- Using the Resource Adapter Deployment Descriptor (also called *ra.xml*).

- Using factory-based configuration. This type of configuration is different from application server vendor-to-application server vendor.

Note: You must choose one of the three ways for Tuxedo JCA Adapter configuration. A combination of the three is not supported.

This section contains the following topics:

- Configuration Overview

- Default Configuration

- dmconfig Configuration

- Resource Adapter Deployment Descriptor-Based Configuration
- Factory-Based Configuration

Configuration Overview

The XML-based configuration file provides a way to configure using complete the capabilities of Tuxedo JCA Adapter. It is suitable for users with complex configuration requirements. To configure using this method, you must configure the `"resourceadapter-class"` with `com.oracle.tuxedo.adapter.TuxedoResourceAdapter` in the `"resourceadapter"` element of the Resource Adapter Deployment Descriptor.

Listing 1 shows a Resource Adapter Deployment Descriptor fragment that enables using an XML-based `dmconfig` file.

Listing 1   Resource Adapter Deployment Descriptor For Using XML-Based Configuration File

```
...
<resourceadapter>
  <resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoResourceAdapter</
resourceadapter-class>
…
    <outbound-resourceadapter>

      <connection-definition>
<managedconnectionfactory-class>com.oracle.tuxedo.adapter.spi.TuxedoManage
dConnectionFactory</managedconnectionfactory-class>
…
  </outbound-resourceadapter>
</resourceadapter>
…
```

The Resource Adapter Deployment Descriptor (commonly known by its file name `ra.xml`) based configuration utilizes the custom properties in the deployment descriptor. It provides an

Tuxedo JCA Adapter configuration capability subset that is suitable for client-side only operations. It provides an easy way to configure an Tuxedo JCA Adapter.

To configure using Resource Adapter Deployment Descriptor method, you must configure the `"resourceadapter-class"` with `com.oracle.tuxedo.adapter.TuxedoClientSideResourceAdapter` in the 'resource adapter' element of the Resource Adapter Deployment Descriptor.

Listing 2 shows a Resource Adapter Deployment Descriptor fragment that enables using Custom Properties-based configuration.

Listing 2   Resource Adapter Deployment Descriptor Using Custom Properties

```
<resourceadapter>

<resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoClientSideResourceA
dapter</resourceadapter-class>
…
    <outbound-resourceadapter>

      <connection-definition>

<managedconnectionfactory-class>com.oracle.tuxedo.adapter.spi.TuxedoManage
dConnectionFactory</managedconnectionfactory-class>
…
  </outbound-resourceadapter>

</resourceadapter>
…
```

The factory-based configuration uses a vendor-specific way to configure connection factories. Each connection factory has its own configuration. It provides a larger subset of configuration capability of Tuxedo JCA Adapter when compares to Resource Adapter Deployment Descriptor-based configuration. It provides an easier way to configure an Tuxedo JCA Adapter.

To configure using factory-based configuration method user must configure the `"resourceadapter-class"` with `com.oracle.tuxedo.adapter.TuxedoFBCResourceAdapter` in the `'resourceadapter'` element and the `"managedconnectionfactory-class"` in the `'connection-definition'` of the `'outbound-resourceadapter'` element with the value `com.oracle.tuxedo.adapter.spi.TuxedoFBCManagedConnectionFactory`.

Listing 3shows a Resource Adapter Deployment Descriptor fragment that enables using factory-based configuration.

Listing 3   Resource Adapter Deployment Descriptor Fragment

```
…
<resourceadapter>

<resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoFBCResourceAdapter<
/resourceadapter-class>
…
  <outbound-resourceadapter>

    <connection-definition>

<managedconnectionfactory-class>com.oracle.tuxedo.adapter.spi.TuxedoFBCMan
agedConnectionFactory</managedconnectionfactory-class>
…
  </outbound-resourceadapter>

</resourceadapter>
```

If the /Domain configuration, `dmconfig` file, is configured in the deployment descriptor file, but the `"resourceadapter-class"` class configured is not `com.oracle.tuxedo.adapter.TuxedoResourceAdapter`, then the `dmconfig` information is to be ignored.

If the `resourceadapter-class` class configured is `TuxedoResourceAdapter`, all resource adapter configuration-related custom properties are ignored.

Default Configuration

The "default" configuration refers to the Tuxedo JCA Adapter ability to dynamically generate configuration for some of the configuration object types if you do not configure any one of these object types explicitly. Not all configuration object types support the "default" configuration. It is available for `Local AccessPoint`, `SessionProfile`, `Session`, and `Import`. The rest of the configuration object types (such as `Resources`, `RemoteAccessPoint`, and `Export`), do not support this type of configuration.

The "default" configuration works with all three configuration styles. It is not necessary to configure `LocalAccessPoint`, `SessionProfile`, `Session`, and `Import` in some situations. In those instances, you may only need to configure `RemoteAccessPoint` in the `dmconfig` file or `remoteAccessPontSpec` in the `Resource Adapter Deployment Descriptor` file, or `remoteAccessPointSpec` in the factory-based configuration file. This greatly enhances Tuxedo JCA Adapter usability.

This section contains the following topics:

- Default LocalAccessPoint

- Default SessionProfile

- Default Session

- Default Import

Default LocalAccessPoint

The `default LocalAccessPoint` does not allow you not to configure a `LocalAccessPoint`. It does not have a listening end point and will not accept inbound connecting requests from Oracle Tuxedo GWTDOMAIN gateway. The Connection Policy can only be "`ON_STARTUP`".

In a `dmconfig`-based configuration and Deployment Descriptor-based configuration, there can be only one `default LocalAccessPoint`; however, for a factory- based configuration each factory can have its own `default LocalAccessPoint`.

The `default LocalAccessPoint`, when used, creates a UUID-based `LocalAccessPointId`. This UUID-based `LocalAccessPointId` is written in a file named `.lapid`, but for factory-based configurations, a file named "`.lapid.<factory-name>`" is created.

In a `dmconfig`-based configuration, the `default LocalAccessPoint` does not support SSL. If you need SSL, a `LocalAccessPoint` must be configured.

In Deployment Descriptor-based configuration the `default LocalAccessPoint` supports SSL in a limited fashion by configuring custom properties `identityKeyStoreFileName`, `privateKeyAlias`, `trustedKeyStoreFileName` in the Resource Adapter Deployment Descriptor. You cannot configure `Key Store`, `Identity Alias`, and `Trusted Certificate Store` with password protection.

In a factory-based configuration, the `default LocalAccessPoint` supports SSL. You can specify a password for Key Store, Identity Alias, and Trusted Certificate Store.

Dynamic RemoteAccessPoint (RAP) Insertion

In order to make `default LocalAccessPoint` to work, Oracle Tuxedo GWTDOMAIN gateway configuration is required in order to make this simplified /Domain configuration to work.

GWTDOMAIN gateway must be modified to allow Dynamic RemoteAccessPoint (RAP) Registration. If `DYNAMIC_RAP` is set to `YES`, it also updates the in-memory database of the status of the connection from dynamically registered RAPs. If the connection from those dynamically registered RAP is lost, then the information about that RAP is removed from the in-memory database.

Note: When Dynamic RemoteAccessPoint Registration/Insertion is enabled in the Oracle Tuxedo GWTDOMAIN gateway, not all the remote access points can connect to it without having been configured in the `REMOTE_DOMAINS` section.

Currently, only Tuxedo JCA Adapter with `default LocalAccessPoint` enabled has the ability to connect to a remote Oracle Tuxedo GWTDOMAIN gateway. When GWTDOMAIN receives a connection request, it checks whether the remote domain is configured. If not, then it checks whether `DYNAMIC_RAP` is set to `YES`. If it is set to `YES`, it checks the message data to determine whether the request came from a legitimate Tuxedo JCA Adapter.

`GWADM` must be modified to process the `DM MIB` correctly to reflect the connection status of dynamically registered RAPs. When the connection from dynamically registered RAP s is lost, their entries in the in memory database is also removed so that the `DM MIB` query can return the connection status correctly.

The dynamically registered RAPs are not added to `/DOMAIN` configuration permanently. Their existence is only known when the Session is established. Their existence is lost when the connection is lost.

The `DM_CONNECTION` Oracle Tuxedo `/Domain DMIB` call returns all the connected dynamically registered `RemoteAccessPoint`. All other dynamically registered `RemoteAccessPoints` that are not connected are not shown.

The `OPENCONNECTION DMIB` request is not supported to connect to those dynamically registered RAPs.

The `CLOSECONNECTION` Oracle Tuxedo `/DMIB` request closes the connection and removes the session from those dynamically registered `RemoteAccessPoint`, and returns its connection status as `'UNKNOWN`.

If the `PERSISTENT_DISCONNECT` type of `CONNECTION_POLICY` is honored and `PERSISTENT_DISCONNECT` is in effect, all connection requests from any RAP (whether they are dynamically or non-dynamically registered), are rejected.

Default SessionProfile

The adapter-wise default session profile is always created whether or not a `SessionProfile` is configured in the dmconfig configuration file. There can only be one default `SessionProfile` for both `dmconfig`-based configuration and Deployment Descriptor-based configuration; it is the adapter-wide default `SessionProfile`.

The adapter-wide default `SessionProfile` cannot be modified when using `dmconfig=`based configuration; however, if you need a different `SessionProfile` other than the default one, you should configure the appropriate `SessionProfile`, and assign it to the target `Session`.

The adapter-wise default `SessionProfile` can be modified when using Resource Deployment Descriptor-based configuration with a set of custom properties. Since there is no specific session profile that can be configured explicitly using a Deployment Descriptor-based configuration, this adapter-wise default `SessionProfile` is used for all Sessions.

The factory-based configuration adds the support for factory-wise default `SessionProfile` in addition to the adapter-wise default `SessionProfile`. If you use this configuration method, you cannot modify the adapter-wise default `SessionProfile`; however, you can modify the factory-wise default `SessionProfile` using a set of factory custom properties. If the default `SessionProfile` is not suitable for any connection factories created connection, then you can configure the factory-wise default `SessionProfile` for each connection factories.

Table 3 lists the default configuration `SessionProfile` type elements.

Note: The `SessionProfile` related properties are configured in the Resource Adapter Deployment Descriptor file shown in Table 3 are used in the construction of the default `SessionProfile`.

Table 3 SessionProfile Type Elements

Element Name	Property Name	Type	Default Value	Description
Security	spSecurity	string	NONE	The type of /Domain session authentication required.
BlockTime	spBlockTime	int	60000	The maximum number of milliseconds allowed for a blocking outbound request using this profile.
Interoperate	spInteroperate	boolean	false	Specifies whether the session is allowed to interoperate with a remote Tuxedo 6.5 release GWTDOMAIN gateway or not.
ConnectionPolicy	N/A	string	ON_STARTUP	The condition under which this GWTDOMAIN session is established.
CredentialPolicy	spCredentialPolicy	string	LOCAL	The user credential propagation policy. When the value is LOCAL, then there is no propagation.
RetryInterval	spRetryInterval	long	60	The number of seconds that a session waits between automatic connection establishment attempts. This is meaningful only when ConnectionPolicy is set to ON_STARTUP.
MaxRetries	spMaxRetries	long	9223372063857758078	The maximum number of times that Tuxedo JCA Adapter tries to establish a session connection to remote Oracle Tuxedo access points. This is meaningful only when ConnectionPolicy is set to ON_STARTUP.

Table 3 SessionProfile Type Elements

Element Name	Property Name	Type	Default Value	Description
CompressionLimit	spCompressionLimit	int	2147483647	The compression threshold a session uses when sending data to a remote Oracle Tuxedo access point. Application buffers larger than this size are compressed.
MinEncryptBits	spMinEncryptBit	string	0	The minimum encryption key length (in bits) a session uses after establishing a session connection. A value of 0 indicates encryption is optional. Value "256" is for SSL. Key strength of 256 bits is for SSL support only.
MaxEncryptBits	spMaxEncryptBit	string	128	The maximum encryption key length (in bits) a session uses to transport data after establishing a session connection. A value of 0 indicates encryption is optional. Value "256" is for SSL. Key strength of 256 bits is for SSL support only.
KeepAlive	spKeepAlive	long	0	Specifies whether a GWTDOMAIN session is configured with application level keep alive, and its maximum idle time before wait timer start ticking. Measured in milliseconds.
KeepAliveWait	spKeepAliveWait	long	10000	Specifies whether a session requires the application level keep alive acknowledgement, and how long it will wait without receiving acknowledgement before declaring the connection inaccessible

Note: The property name is the same for both Resource Adapter Deployment Descriptor-based configuration and factory-based configuration.

Default Session

If Resource Adapter Deployment Descriptor-based configuration or factory-based configuration is used, or there is no `Session` configured in the `dmconfig` file, the session is implicitly created between all *local* access points and all *remote* access points. This is called default Session. If default Session is used, it can only use the adapter-wise default `SessionProfile` when Resource Adapter Deployment Descriptor-based or `dmconfig`-based configuration is used. The factory-based configuration uses the factory-wise default `SessionProfile`. Any factory not configured with its own default `SessionProfile` uses the adapter-wise default `SessionProfile`.

For example, if two `RemoteAccessPoint` are configured and the `default LocalAccessPoint` is used and there is no Session configured, then two default Session are created: one to the first `RemoteAccessPoint` configured, and one to the second `RemoteAccessPoint` configured. Both sessions use default Session Profile

Default Import

Tuxedo JCA Adapter allows you to access remote Oracle Tuxedo services or resources through the configured sessions even when there is no Oracle Tuxedo service or resource configured. This feature is called Default Import. If there is at least one configured Oracle Tuxedo service or resource, this feature is automatically disabled and only the requests that target configured Oracle Tuxedo services or resources can be forwarded to Oracle Tuxedo.

When there is no Oracle Tuxedo service or resource configured as Import in the Tuxedo JCA Adapter configuration, request filtering is not performed and all requests are forwarded to Oracle Tuxedo using the name specified in the service request invocation. If more than one session is configured or created implicitly by the Tuxedo JCA Adapter, all the service requests are load-balanced among those sessions using a `RoundRobin` algorithm.

When default Import is enabled, all Oracle Tuxedo services/resources can be accessed by the Tuxedo JCA Adapter through all the configured Sessions or implicitly created default Sessions to any connected Oracle Tuxedo application domain.

dmconfig Configuration

The Tuxedo JCA Adapter configuration file is an XML-based file represented by a property named '`dmconfig`' in the Resource Adapter Deployment Descriptor (`ra.xml`). This property

value can be either an absolute path to a configuration file, or it can be represented as a resource of the resource `archive` (RAR) file. You must use this method for configuration when full blown client and server operations are required.

Listing 4 shows an example Tuxedo JCA Adapter using a 'dmconfig' file. In this example, the full path name to the configuration file is: `/home/work/adapter/dmconfig.xml`.

Listing 4 dmconfig Full Path Example

```
...
<resourceadapter>
  <resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoResourceAdapter</
   resourceadapter-class>
  <config-property>
    <config-property-name>dmconfig</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>/home/work/adapter/dmconfig.xml</config-proper
      ty-value>
  <config-property>
...
```

Listing 5 shows an example telling the Tuxedo JCA Adapter that it uses a 'dmconfig' file and is packaged as part of the resource archive with the resource file name 'dmconfig.xml'. However, if this configuration resource file is not found in the archive, it is treated as a configuration file located in the current working directory.

Listing 5 dmconfig Archive

```
...
<resourceadapter>
  <resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoResourceAdapter<
   /resourceadapter-class>
<config-property>
  <config-property-name>dmconfig</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>dmconfig.xml</config-property-value>
```

```
<config-property>

...
```

TuxedoConnector Root Element

There is only one root element, "TuxedoConnector", in the Tuxedo JCA Adapter configuration file. It is represented by the complex type TuxedoConnectorType. It contains the following elements (listed in Table 4):

- Resources

- Local Access Point

- Remote Access Point

- Session Profile

- Session

- Import

- Export

Table 4 TuxedoConnectorType Element

Element Name	Type	Occurrence	Description
Resources	ResourceType	0..1	Tuxedo JCA Adapter environment
LocalAccessPoint	LocalAccessPointType	0..unbounded	Local access point
RemoteAccessPoint	RemoteAccessPointType	0..unbounded	Remote access point
SessionProfile	SessionProfileType	0..unbounded	Session profile including QoS configuration information.
Session	SessionType	0..unbounded	Defines the possible session/connection to a remote domain.

Table 4 TuxedoConnectorType Element

Element Name	Type	Occurrence	Description
`Import`	`ImportType`	`0..unbounded`	Defines a remote resource available to the adapter
`Export`	`ExportType`	`0..unbounded`	Defines a local resource available to remote domain.

The configuration must use the `<TuxedoConnector>` and `</TuxedoConnector>` tags as shown in Listing 6.

Listing 6 TuxedoConnector Tags Example

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

Resources

The `Resources` element is represented by `ResourceType`. It specifies the resource adapter execution environment.Only one `Resource` element can be configured in the Tuxedo JCA Adapter configuration file. Table 5 lists the "`ResourceType`" elements.

Table 5 ResourceType Element

Element Name	Type	Occurrence	Description
`FieldTable16Classes`	`string`	`0..unbounded`	Fully qualified field table 16 classes for FML.
`FieldTable32Classes`	`string`	`0..unbounded`	Fully qualified field table 32 classes for FML32.
`ViewFile16Classes`	`string`	`0..unbounded`	Fully qualified VIEW table 16 classes for VIEW.

Table 5 ResourceType Element

Element Name	Type	Occurrence	Description
ViewFile32Cla sses	string	0..unbounded	Fully qualified VIEW table 32 classes for VIEW32.
ApplicationPa sswordEncrypt ed	string	0..1	The application password for joining an Oracle Tuxedo application. The password is encrypted. The password length cannot exceed 30 characters and should be the same as Oracle Tuxedo.
TpusrFile	string	0..1	Multibyte character set encoding name.
RemoteMBEncod ing	string	0..1	The TPUSR file full path name.
MBEncodingMap File	string	0..1	Encoding map file full path name.

Note: There is no attribute defined for any "ResourceType" element; however, the "TPUsrFile" element can be overridden by the TpusrFile element in the RemoteAccessPoint.

Listing 7 shows a "Resources" configuration example.

Listing 7 Resources Configuration Example

```
<Resources>
  <FieldTable16Classes>tuxedo.test.fml16.FieldTbl16</FieldTable16Classes>
  <ViewFile32Classes>tuxedo.test.simpapp.View32</ViewFile32Classes>
  <ApplicationPasswordEncrypted>tuxpassword</ApplicationPasswordEncrypted>
</Resources>
```

If "ApplicationPassowrdEncrypted" is configured, it is required to run the provided utility com.oracle.tuxedo.tools.DMConfigChecker to encrypt the password, and optionally generate a key store. In this case, the keyFileName must point to a valid key store file or key store resource. If keyFileName is not configured in the Resource Adapter Deployment Descriptor, the Tuxedo JCA Adapter fails to decrypt the password.

Listing 8 shows an example that tells the Tuxedo JCA Adapter that a key store resource is configured.

Listing 8   Key Store Resource Example

```
...
<resourceadapter>
  <resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoResourceAdapter</
   resourceadapter-class>
  <config-property>
     <config-property-name>dmconfig</config-property-name>
     <config-property-type>java.lang.String</config-property-type>
     <config-property-value>dmconfig.xml</config-property-value>
  <config-property>
  <config-property>
     <config-property-name>keyFileName</config-property-name>
     <config-property-type>java.lang.String</config-property-type>
     <config-property-value>foo.key</config-property-value>
  <config-property>
...
```

Local Access Point

The Local Access Point element is represented by `LocalAccessPointType`. It specifies a listening address and possible link-level failover address. Table 6 lists the `LocalAccessPointType` elements.

Table 6 LocalAccessPointType Element

Element Name	Type	Occurrence	Description
AccessPointId	string	0..1	The connection principal name. It must be *globally* unique.
			Note: Globally unique means the identification specified in the configuration must be unique within all the interconnected Oracle Tuxedo Domains and Application Servers through Oracle Tuxedo GWTDOMAIN gateway, WebLogic Server WTC, and Tuxedo JCA Adapter.
			If not specified, it uses the *locally* unique LocalAccessPoint "name" attribute value (see Listing 9).
NetworkAddress	string	1..unbounded	The local access point listening address including both host address and port number. Specify the TCP/IP address in the format //hostname:port or //#.#.#.#:port.
SSLInfo	complexType	0..unbounded	Specifies whether SSL encryption is used to ensure communication or not. If not configured, LLE is used.

The SSLInfo element is an anonymous complex type. If included in the configuration, all its elements must be configured, and SSL is used as the transport mechanism. If it is not included, TCP/IP is used as the transport mechanism and uses link-level encryption for data privacy if encryption is required. Table 7 lists SSLInfo elements.

Table 7 SSLInfo Elements

Element Name	Type	Occurrence	Description
`MutualAuthent icationRequir ed`	`boolean`	`0..1`	Specifies if client authentication is required for SSL communication. The default value is `"false"`.
`IdentityKeySt oreFileName`	`string`	`1..1`	Full identity key store file path name.
`IdentityKeySt orePassPhrase Encrypted`	`string`	`1..1`	Password used to encrypt the identity key store.
`PrivateKeyAli as`	`string`	`1..1`	Alias in the identity key store used to retrieve private key.
`PrivateKeyPas sPhraseEncryp ted`	`string`	`1..1`	Encrypted password used to decrypt the private key in the identity key store.
`TrustKeyStore FileName`	`string`	`1..1`	Full trust key store file path name.
`TrustKeyStore PassPhraseEnc rypted`	`string`	`1..1`	Pass phrase used to decrypt the trust key store when retrieving certificates.

The "`name`" attribute is defined for `LocalAccessPointType`. It is used to identify the configuration record as represented by `LocalAccessPointType`. It specifies a *locally* unique local access point name as shown in Listing 9.

Listing 9 LocalAccessPoint Name Attribute

```
<LocalAccessPoint name="LDOM1">
  <LocalAccessPointId>Godfried</LocalAccessPointId>
 <NetworkAddress>//neocortex:14001</NetworkAddress>
</LocalAccessPoint>
```

'LDOM1' must be locally unique. 'Gotfried' must be globally unique. If the 'AccessPointId' element is not specified, the value of the name attribute is used as 'AccessPointId'. In this scenario, the value of the name attribute must be globally unique.

By default, SSL only authenticates the server (connection request responder). However, to enable the client authentication the "MutualAuthenticationRequired" element must be set to "true."

Remote Access Point

The Remote Access Point element is represented by RemoteAccessPointType. It defines a network address for remote Oracle Tuxedo Domain access points. Table 8 lists RemoteAccessPointType elements.

Table 8 RemoteAccessPointType Element

Element Name	Type	Occurrence	Description
AccessPointId	string	0..1	It is used by default as connection principal name that is used to identify this remote access point when attempting to establish a session with remote Tuxedo access point. It must be *globally* unique.
			Note: Globally unique means the identification specified in the configuration must be unique within all the interconnected Oracle Tuxedo Domains and Application Servers through Oracle Tuxedo GWTDOMAIN gateway, WebLogic Server WTC, and Tuxedo JCA Adapter
			If not specified, it uses the *locally* unique RemoteAccessPoint "name" attribute value (see Listing 10).
NetworkAddress	string	1..unbounded	The host network address and port number of the remote Oracle Tuxedo access point. Specify the TCP/IP address in the format //hostname:port or //#.#.#.#:port.
TpusrFile	string	0..1	The full path name to a TPUSR file for this remote access point.

Table 8 RemoteAccessPointType Element

Element Name	Type	Occurrence	Description
AllowAnonymous	boolean	0..1 Value set {true, false}	Indicates whether the remote Oracle Tuxedo access point allows anonymous access or not. The default value is "false".
DefaultApplicationKey	string	0..1	The default application key value for the remote Oracle Tuxedo access point. If not specified, then "-1" is assumed.
CustomApplicationKey	string	0..1	Configures the custom application key generator. If not specified, the default application key generator is used.

The `CustomApplicationKey` element is an anonymous complex type. If not specified, the default Application Key plug-in is used. If specified, all of its elements must be configured and the `CustomApplicationKey` plug-in class is loaded for every session that communicates with this remote access point. Table 9 lists `CustomApplicationKey` elements.

Table 9 CustomApplicationKey Element

Element Name	Type	Occurrence	Description
ApplicationKeyClass	string	1..1	The fully qualified application key generator class name.
ApplicationKeyClassParam	string	1..1	The parameter string passed to the application key generator when it is initialized at runtime

The "name" attribute is defined for `RemoteAccessPoint`. It specifies the *locally* unique Remote Access Point Name. Listing 10 shows a `RemoteAccessPointType name` attribute example.

Listing 10 RemoteAccessPoint Name Attribute Example

```
<RemoteAccessPoint name="RDOM1">
  <AccessPointId>Geneve</AccessPointId>
  <NetworkAddress>//bluestar:11023</NetworkAddress>
```

```
    <TpusrFile>/tja/lady-geneve/tpusr</TpusrFile>
</RemoteAccessPoint>
```

'RDOM1' must be locally unique. 'Geneve' must be globally unique. If the 'AccessPointId' element is not specified, the value of the name attribute is used as 'AccessPointId'. In this scenario, the value of the name attribute must be globally unique.

Session Profile

The Session Profile element is represented by SessionProfileType. It contains all the QoS parameters for TDOMAIN sessions between an Tuxedo JCA Adapter Local Access Point and an Oracle Tuxedo Remote Access Point. Table 10 lists SessionProfileType elements.

Table 10 SessionProfileType Element

Element Name	Type	Occurrence	Description
Security	string	0..1 Value set {NONE, APP_PW, DM_PW}	The type of /Domain authentication required. The default value is NONE.
BlockTime	int	0..1 Value range 0..2147483647	The maximum number of milliseconds allowed for a blocking outbound request using this profile. The default value is 60000.
Interoperate	boolean	0..1 Value set {true, false}	Specifies whether the session is allowed to interoperate with a remote Tuxedo 6.5 release GWTDOMAIN gateway or not. The default value is "false".
ConnectionPolicy	string	0..1 Value set {ON_DEMAND, ON_STARTUP, INCOMING_ONLY}	The condition under which this GWTDOMAIN session is established. The default value is ON_DEMAND.
ACLPolicy	string	0..1 Value set {LOCAL, GLOBAL}	The ACL policy to be enforced on this GWTDOMAIN session. The default value is LOCAL.

Table 10 SessionProfileType Element

Element Name	Type	Occurrence	Description
CredentialPolicy	string	0..1 Value set {LOCAL, GLOBAL}	The user credential propagation policy. When the value is Local, then there is no propagation. The default value is LOCAL.
RetryInterval	long	0..1 Value range 0..2147483647	The number of seconds that sessions wait between automatic connection establishment attempts. Set this element value only when ConnectionPolicy is set to ON_STARTUP.
MaxRetries	long	0..1 Value range 0..922372063857758	The maximum number of times that the Tuxedo JCA Adapter tries to establish a session connection to a remote Oracle Tuxedo access point before it stops trying. Set this element only when ConnectionPolicy is set to ON_STARTUP. Default value is 922337206385775807.
CompressionLimit	int	0..1 Value range 0..2147483647	The compression threshold a session uses when sending data to a remote Oracle Tuxedo Access Point. Application buffer larger than this size are compressed. Default value is 2147483647.
MinEncryptBit	string	0..1 Value set {"0", "40", "56", "128", "256"}	The minimum encryption key length (in bits) a session uses when establishing a session connection. A value of 0 tells the Tuxedo JCA Adapter that the session created using this profile allows no encryption. Key strength of 256 bits is for SSL support only. Default value is 0.

Table 10 SessionProfileType Element

Element Name	Type	Occurrence	Description
MaxEncryptBit	string	0..1 Value set {"0", "40", "56", "128", "256"}	The maximum encryption key length (in bits) a session uses when establishing a session connection. A value of 0 tells the Tuxedo JCA Adapter that the session created using this profile does not require encryption. Key strength of 256 bits is for SSL support only. Default value is 128. The value specified must be larger or equal to the value specified in MinEncryptBit
KeepAlive	long	0..1 Value range {0..2147483647}	Specifies if a GWTDOMAIN session is configured with Application Level Keep Alive, and its maximum idle time before the wait timer start s. Measured in milliseconds. Default value is 0 (which means the KeepAlive feature is disabled).
KeepAliveWait	long	0..1 Value range {0..2147483647}	Specifies if a session requires the Application Level Keep Alive acknowledgement, and how long it will wait without receiving acknowledgement before declaring the connection inaccessible. Measured in milliseconds.Default value is 0 (which means ignore the KeepAlive acknowledgement; however, if the KeepAlive feature is disabled, the Tuxedo JCA Adapter ignores this part of the configuration information when a session is created using this profile).

The "name" attribute is defined for SessionProfileType. It is used by the Session object to get the correct session profile. Listing 11 shows a SessionProfileType name attribute example.

Listing 11   SessionProfile Name Attribute Example

```
<SessionProfile name="profile1">
  <Security>DM_PW</Security>
```

```
    <ConnectionPolicy>ON_STARTUP</ConnectionPolicy>
    <ACLPolicy>Global</ACLPolicy>
    <CredentialPolicy>Global</CredentialPolicy>
    <RetryInterval>100</RetryInterval>
</SessionProfile>
```

Session

The Session element is represented by "`SessionType`". It specifies a permissible connection between a Local Access Point and a Remote Access Point. Only one session can be configured between a Local Access Point and a Remote Access Point. Table 11 lists the `SessionType` elements.

Table 11 SessionType Elements

Element Name	Type	Occurrence	Description
LocalAccessPo intName	string	1..1	The local access point that is used to compose a TDOMAIN session. This "`LocalAccessPoint`" refers to the "`name`" attribute of a "`LocalAccessPoint`" element.
RemoteAccessP ointName	string	1..1	The remote access point that is used to compose a TDOMAIN session. This "`LocalAccessPoint`" refers to the "`name`" attribute of a "`LocalAccessPoint`" element.
ProfileName	string	0..1	The profile to be used for a session. If not specified uses adapter-wise default session profile.
PasswordPair	complex Type	0..2	The password pair is used when SECURITY equals DM_PW to authenticate the session.

The `PasswordPair` element is an anonymous complex type. At most, two password pairs can be configured. It allows you to configure passwords for Oracle Tuxedo Domain Session Authentication. Table 12 lists the `PasswordPair` elements.

Table 12 PasswordPair Element

Element Name	Type	Occurrence	Description
LocalPassword Encrypted	string	1..1	The encrypted local password. The password length cannot exceed 30 characters.
RemotePasswor dEncrypted	string	1..1	The encrypted remote password. The password length cannot exceed 30 characters.
ActivationTim e	string	0..1	The date and time string used to indicate when a password pair becomes effective. If not specified, then it is assumed already become effective. The format is YYYY:MM:DD:hh:mm:ss.
DeactivationT ime	string	0..1	The date and time string used to indicate when a password pair becomes obsolete. If not specified, it is assumed that it will never expire. Same format as ActivationTime.

The "name" attribute is defined for SessionType. It is used to identify a TDOMAIN session. Listing 12 shows a SessionType name attribute example.

Listing 12 SessionType Name Example

```
<Session name="session1_1">
  <LocalAccessPointName>LDOM1</LocalAccessPointName>
  <RemoteAccessPointName>RDOM1</RemoteAccessPointName>
  <ProfileName>profile1</ProfileName>
</Session>
```

If no Session is configured, by default the Tuxedo JCA Adapter creates sessions between all Local Access Points and all Remote Access Points. These dynamically created sessions can only use the default Session Profile.

Import

The Import element is represented by `ImportType`. It identifies an existing remote Oracle Tuxedo Application Domain resource that can be accessed by the Tuxedo JCA Adapter client. Table 13 lists `ImportType` elements.

Table 13 ImportType Element

Element Name	Type	Occurrence	Description
RemoteName	string	0..1	The actual remote Oracle Tuxedo resource name being exported by the Oracle Tuxedo TDomain gateway. If not specified, it has the same value as the "`name`" attribute.
SessionName	string	1..unbounded	The name of the session that imports a resource from the remote Oracle Tuxedo application domain.
LoadBalancing	string	0..1 Value set {RoundRobin, Random}	The load balancing algorithm used for an imported resource. The default value is `RoundRobin`.

There are three defined attributes: `name`, `autotran`, and `trantime`.

- `name`

  Specifies the resource name to used by a JATMI `tpcall()` as service name or by a CCI `execute()` as function name.

- `autotran`

  Enables/disables of `AUTOTRAN`. It only accepts `true` or `false` values. If set to true, when this imported resource is invoked by an Tuxedo JCA Adapter client outside a global or local transaction the Tuxedo JCA Adapter starts a transaction with a remote Oracle Tuxedo Domain.

  If it is set to false, the Oracle Tuxedo JCA does not start a transaction if the client service request is outside of global or local transaction. However, setting it to false does not prevent client service requests of this remote resource to participate in global or local transactions.

  If it is not configured then the adapter-wise `AUTOTRAN` property in the Resource Adapter Deployment Descriptor is used to determine the `AUTOTRAN`.

- `trantime`

  Specifies the transaction timeout value for AUTOTRAN of this resource. The value is measured in seconds. If not specified and AUTOTRAN is required then the 'appManagedLocalTxTimeout' property of the Resource Adapter Deployment Descriptor is used.

  If `appManagedLocalTxTimeout` is not specified, the JVM property `com.oracle.tuxedo.adapter.appManagedLocalTxTimeout` is used.

  If `com.oracle.tuxedo.adapter.appManagedLocalTxTime` JVM property is not specified, it defaults to 300 seconds.

  Listing 13 shows an `ImportType name` example that describes an imported resource with name TUXUPPER. The AUTOTRAN transaction timeout is set to 10 seconds

Listing 13   ImportType Name Example

```
<Import name="TUXUPPER" autotran="true" trantime="10">
   <RemoteName>TOUPPER_1</RemoteName>
   <SessionName>session1</SessionName>
   <LocalBalancing>RoundRobin</LoadBalancing>
</Import>
```

Export

The Export element is represented by `ExportType`. It specifies a local resource that is accessible from a remote Oracle Tuxedo Application Domain. Table 14 lists `ExportType` elements.

Table 14   ExportType Element

Element Name	Type	Occurrence	Description
RemoteName	string	0..1	Resource name the Oracle Tuxedo application uses to access service in an application server. If not specified, it has the same value as the name attribute.
SessionName	string	1..unbounded	The session that is allowed to access to this local resource.

Table 14 ExportType Element

Element Name	Type	Occurrence	Description

`Type`	`string`	`0..1`
		`Value set`
		`{EJB,POJO,MD B}`
			Type of resource. The default is `EJB`.

`Source`	`string`	`1..1`	This is the JNDI name for EJB, for instance `tuxedo.services.TolowerEJBHome`; or the target class of the POJO, for instance, `com.abc.test.MyTolower`; or the JNDI name for MDB, for instance is/echo. It must be specified.
`SourceLocatio n`	`string`	`0..1`	This is the target jar for POJO. Its value is ignored if export type is EJB or MDB.

The `"name"` attribute is defined for `ExportType`. It is used to identify an exported resource. Listing 14 shows an `ExportType name` example.

Listing 14 ExportType Name Example

```
<Export name="tolower">
   <SessionName>session1</SessionName>
   <RemoteName>wtolower</RemoteName>
   <Type>EJB</Type>
   <Source>tuxedo.services.TolowerEJBHome</Source>
</Export>
```

Resource Adapter Deployment Descriptor-Based Configuration

The main component used to deploy and repackage the resource archive for deployment is the resource adaptor deployment descriptor (the `ra.xml` file in the `META-INF` directory). The Resource Adapter Deployment Descriptor must be configured before repacking the resources into a resource archive.

This XML-based text file can be modified by using a text or XML editor. The downloaded Tuxedo JCA Adapter contains a simple version of the deployment descriptor to assist in configuring the Tuxedo JCA Adapter deployment.

For Deployment Descriptor-based configuration, you must specify an Tuxedo JCA Adapter deployment configuration using the deployment descriptor file (`ra.xml`). There is a set of custom properties to allow you to specify the configuration. The scope of all properties specified this way is adapter-wise.

Resource Adapter Deployment Descriptor Properties

The deployment descriptor-based configuration style is based on standard simple property types: `config-property-name`, `config-property-type`, and `config-property-value`. These property types cannot be repeated. They are available in the resource adapter section of the Resource Adapter Deployment Descriptor "`resourceadapter`" file.

The Resource Adapter Deployment Descriptor configuration method supports the following types of properties:

- Resource Related Properties
- Local Access Point Related Properties
- RemoteAccessPoint Related Properties
- SessionProfile Related Properties
- Import Related Property

Resource Related Properties

Most of the Resources elements specified in the `dmconfig` file are available for deployment descriptor-based configuration.

Table 15 lists the Resource Adapter Deployment Descriptor `Resources` properties.

Table 15 Resources Properties

Property	Type	Default Value	Description
fieldTable16 Classes	String	None	Field table 16 classes for FML. It is a comma-separated list.
fieldTable32 Classes	String	None	Field table 32 classes for FML32. It is a comma-separated list.
viewFile16Cl asses	String	None	VIEW Table 16 classes for VIEW. It is a comma-separated list.
viewFile32Cl asses	String	None	VIEW Table 32 classes for VIEW32. It is a comma-separated list.
tpusrFile	String	None	The full path name to the TPUSR file.
remoteMBEnco ding	String	None	The Multi-Byte encoding used by Oracle Tuxedo.
mBEncodingMa pFile	String	None	Full path name to the encoding map file.

Listing 15 shows a configuration example that describes two VIEW32 classes information using Resource Adapter Deployment Descriptor-based custom property configuration.

Note: This causes the Tuxedo JCA Adapter to use the viewFile32Classes with full package name accessible through SYSTEM CLASSPATH.

Listing 15   ra.xml File VIEW32 Custom Property Configuration

```
…
<config-property>
  <config-property-name>viewFile32Classes<config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>tuxedo.view32.view1,tuxedo.view32.view2
   </config-property-value>
</config-property>
…
```

Local Access Point Related Properties

The majority of the `LocalAccessPoint` type elements are not available in a resource adapter descriptor-based configuration; however, a single `localAccessPointSpec` can be specified.

Table 16 lists the Resource Adapter Deployment Descriptor `LocalAccessPoint` properties.

Table 16 LocalAccessPoint Properties

Property	Type	Default Value	Description
`localAccessPointSpec`	String	None	The syntax is `//<network address>:<port>/domainId=<domain id>` By default, the domain id is used as connection principal name. It has to be globally unique.
`identityKeyStoreFileName`	String	None	The full path name of the identity key store file name.
`privateKeyAlias`	String	None	The alias in the identity key store to be used to retrieve private key.
`trustedKeyStoreFileName`	String	None	The full path name of the trusted key store file name.

The `localAccessPointSpec` property is optional. When specified in a non-clustered environment then it is useful if you want to have all configuration information in the Resource Adapter Deployment Descriptor file. However, when it is specified in a clustered environment where the configuration is copied to all cluster nodes, all Tuxedo JCA Adapters have the same access point identification.In this situation, the connection behavior becomes unpredictable and is not supported. It can only use default Session, and default `SessionProfile`.

To get around this clustered environment problem, do not to configure it at all (then the `default LocalAccessPoint` is created with UUID-based `LocalAccessPointId`). This UUID-based `LocalAccessPointId` is written in a file named `.lapid` in the current working directory.

Listing 16 provides an `localAccessPointSpec` custom configuration example.

Listing 16   LocalAccessPoint Custom Property Configuration Example

```
<config-property>
   <config-property-name>localAccessPointSpec</config-property-name>
   <config-property-type>java.lang.String<config-property-type>
   <config-property-value>//localhost:12345/domainId=jdom_id
   </config-property-value>
</config-property>
```

This example shows a `LocalAccessPoint` with `AccessPointName` and `AccessPointId` equal to `jdom_id` is created and listens for the incoming connection requests at `port12345` on the local host.

If the `default LocalAccessPoint` is created, the connection policy can only be `"ON_STARTUP"`. The listening endpoint is not created.

When `"default LocalAccessPoint"` is constructed by the Tuxedo JCA Adapter dynamically, there is no listening endpoint and it is for Client-Side only operation mode. There is no incoming connection request possible.

SSL/TLS can also be supported through the Resource Adapter Deployment Descriptor using the three properties related to identity and key stores. However, there are two limitations:

- It only supports identity and trusted key stores without password protection;

- All the Tuxedo JCA Adapter instances in a cluster will have the same values.

If this is not desirable, you must either modify the Resource Adapter Deployment Descriptor for all the nodes that requires different behavior or change to use `dmconfig` method to configure the clustered Tuxedo JCA Adapter.

RemoteAccessPoint Related Properties

`RemoteAccessPoint` is represented by a text string that contains both networking address and per `RemoteAccessPoint` access control related information; however, in a Resource Adapter Deployment Descriptor property-based configuration, it can be represented by a single `remoteAccessPointSpec` property. Table 17 lists the Resource Adapter Deployment Descriptor `RemoteAccessPoint related` properties.

Table 17 RemoteAccessPoint Properties

Property	Type	Default Value	Description
remoteAccessPointSpec	String	None	This contains both `NetworkAddress` and `AccessPointId` plus the name attribute. This is a comma-separated list of `RemoteAccessPoint`. The `domainId` in each entry is used to replace `AccessPointId`.
			This `domainId` is used by default as connection principal name that is used to identify a remote access point when attempting to establish a session with remote Tuxedo access point. If not specified then it is an error in Deployment Descriptor-based configuration; and the name has to be globally unique.
rapAllowAnonymous	Boolean	{true, false}	Indicates whether the remote Tuxedo access point allow anonymous access or not. The default value is "false".
rapDefaultApplicationKey	String	Any valid Tuxedo application key.	The default application key value for the Oracle Tuxedo access point. If not specified then "-1" is assumed.
rapApplicationKeyClass	String	None	The fully qualified class name of the custom application key generator. If not specified then the default application key generator will be used.
rapApplicationKeyClassParam	String	None	The parameter string passed to the custom application key generator when the class is initialized at runtime.

The `remoteAccessPointSpec` property is a comma-delimited list; a comma separates each remote access point. Each remote access point is represented in a specific format.

If property `rapApplicationKeyClass` is not specified, `rapApplicationKeyClassParam` is ignored if one is configured.

To support link level failover parenthesis is used to group failover addresses together for a `RemoteAccessPoint`. The first address specified in a group is the primary address. The second address in a group backs up the first address, and the third address in a group backs up the second address and so on.

Listing 17 provides an example that configures two `RemoteAccessPoint`. The first is accessible through domain `guinevre` with network address `//bluestar:11023`. The second is accessible through domain `galahad` with network address `//orion:37456`.

Listing 17   RemoteAccessPoint Custom Property Configuration Example

```
<config-property>
  <cofig-property-name>remoteAccessPointSpec</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>//bluestar:11023/domainId=guinevre,//orion:37456/
   domainId=galahad</config-property-value>
</config-property>
```

Both remote domains `guinevre` and `galahad` have the same QoS associated with them. In this case, if `rapApplicationKeyClass` and `rapApplicationKeyClassParam` are specified, they are treated as if they are available to both `RemoteAccessPoint` `guinevre` and `galahad`. You must make `rapApplicationKeyClass` available to both RAP with same fully qualified class path, otherwise the Tuxedo JCA Adapter fails to start.

Listing 18 provides an example that configures two `RemoteAccessPoint` with a failover address. The first is accessible through domain `guinevre` with the primary network address `//bluestar:11023`, and the back up network address `//orion:12345`.

The second is accessible through domain `galahad` with network address `//orion:37456`, and the back up network address `//bluestar:37456`.

Listing 18   RemoteAccessPoint Custom Property Configuration Example

```
<config-property>
  <config-property-name>remoteAccessPointSpec</config-porperty-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>(//bluestar:11023,//orion:12345)/domainId=guinevr
```

```
   e,(//orion:37456,//bluestar:37456)/domainId=galahad
   </config-property-value>
</config-property>
```

There can be only one `remoteAccessPointSpec` property specified in the Resource Adapter Deployment Descriptor file. If there are more than one configured, the application server's JCA container will only honor the last one configured.

There is no default `RemoteAccessPoint`.If `remoteAccessPointSpec` property is not configured, there is no dynamically created `RemoteAccessPoint`.This makes Tuxedo JCA Adapter useless even though it still can be started.

To configure `RemoteAccessPoint` through `remoteAccessPointSpec` property, the `"resourceadapter-class"` element in the Resource Adapter Deployment Descriptor file must be configured using `com.oracle.tuxedo.adapter.TuxedoClientSideResourceAdapter` class.

SessionProfile Related Properties

The information contained in Session Profile can be represented by one set of configuration properties. It contains all the QoS parameters for a TDOMAIN session between a Tuxedo JCA Adapter Local Access Point and an Oracle Tuxedo Remote Access Point.

Table 18 lists the Resource Adapter Deployment Descriptor `SessionProfile` properties.

Table 18 SessionProfile Properties

Property	Type	Default Value	Description
spBlockTime	Integer	0..214748364 7	The maximum number of milliseconds allowed for a blocking outbound request using this profile. The default value is `60000` milliseconds. This can be overridden by the CCI `InteractionSpec.setExecutionTi me()` method call.
spInteropera te	Boolean	{true, false}	Specify whether this session is allowed to interoperate with a remote Tuxedo 6.5 release GWTDOMAIN gateway or not. The default value is `"false"`.

Table 18 SessionProfile Properties

Property	Type	Default Value	Description
spCredential Policy	String	{LOCAL, GLOBAL}	The user credential propagation policy. When set to "LOCAL", there is no propagation. Default value is "LOCAL".
spRetryInter val	Long	0..214748364 7	The number of seconds that the session waits between automatic connection establishment attempts. Default value is 60.
spMaxRetries	Long	0..922337206 385775807	The maximum number of times that this sessions tries to establish a session connection to remote Oracle Tuxedo access point. Default value is 922337206385775807
spCompressio nLimit	Integer	0..214748364 7	The compression threshold all the sessions use when sending user data to a remote Oracle Tuxedo access point. Application buffers larger than this size are compressed, The default value is 21474483647.
spMinEncrypt Bits	String	{"0","40","5 6","128","25 6"}	The minimum encryption key length in bits all the session use after establishing a session connection. A value of 0 indicates encryption may not required. The value "256" is for SSL type of connection The default value is "0".
spMaxEncrypt Bits	String	{"0","40","5 6","128","25 6"}	The maximum encryption key length in bits all the sessions use after establishing a session connection. A value of 0 indicates no encryption is required. The value "256" is for SSL type of connection The default value is "128"

Table 18 SessionProfile Properties

Property	Type	Default Value	Description
spKeepAlive	Long	{0..21474836 47}	Specifies whether all the sessions are configured with Application Level Keep Alive, and its maximum idle time before wait timer start ticking. Default value is "0", it means disable Application Level Keep Alive. The measurement is in milliseconds, and rounded up to the nearest second at runtime.
			When the connection is busy there is no need to send special keep alive message to remote gateway; however, when there is no activities over the connection for the specified amount of time, the a special keep alive message is sent and a timer of spKeepAliveWait, also rounded up to second, will be started. If no acknowledgement is received during this "wait" time, the connection is declared dead and is closed, the session is terminated.
spKeepAliveW ait	Long	{0..21474836 47}	Specifies whether this session requires the acknowledgement of Application Level Keep Alive or not, and how long it will wait without receiving acknowledgement before declaring the connection is inaccessible. The default value is 10 seconds. The measurement is in milliseconds. A value of 0 disables wait timer.

Listing 19 provides a SessionProfile configuration example.

Listing 19 SessionProfile Custom Properties Configuration Example

```
< config-property>
  <config-property-name>spBlockTime</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>
```

```
    <config-property-value>120000</config-property-value>
</config-property>
```

All `SessionProfile` related property configured in the Resource Adapter Deployment Descriptor file is used in the construction of the default `SessionProfile`.

Import Related Property

The Resource Adapter Deployment Descriptor file-based configuration utilizes the Default Import; however, it also provides ability to restrict what can be accessed from adapter to remote Tuxedo application domain in a uniformly way. A single `"impResourceName"` property can be specified and it contains a comma-separated list of remote Oracle Tuxedo services/resources allowed to access.One restriction applies is that these will be applied to all the sessions that are possible.

The `LoadBalancing` algorithm is preset to `RoundRobin` and cannot be changed. If `impResourceName` is specified then there is no default Import created by Tuxedo JCA Adapter.

Table 19 lists the Resource Adapter Deployment Descriptor Import Related property.

Table 19 Import Related Property

Property	Type	Default Value	Description
impResourceN ame	String	Any valid Tuxedo resource name in a comma-separa ted list.	The valid Oracle Tuxedo service name or queue name. There is no name transaction done for the wire protocol. This is basically the same as the `RemoteName` attribute of an Import in the `dmconfig` file.

Listing 20 provides an `impResourceName` example that limits the available remote Oracle Tuxedo resources to `Toupper_1`, and `ECHO`. This property is limited to the Resource Adapter Deployment Descriptor-based configuration; a `"resourceadapter-class"` must be configured with `TuxedoClientSideResourceAdapter` class.

Listing 20   impResourceName Example

```
<config-property>
 <config-property-name>impResourceName</config-property-name>
 <config-property-type>java.lang.String</config-property-name>
 <config-property-value>TOUPPER_1,ECHO</config-property-value>
</config-property>
```

Session

There is no Session related property defined in the Resource Adapter Deployment Descriptor custom property configuration method. If a Resource Adapter Deployment Descriptor-based configuration is used or there is no Session configured in the dmconfig file, a session is implicitly created between all local access points and all the remote access points. This is called a "default Session." If a default Session is used, it can only use the default `SessionProfile`.

For example, if there are two `RemoteAccessPoint` elements configured and the `default LocalAccessPoint` is used and there is no Session configured, two default Sessions are created.

Export

There is no Export-related property defined in the Resource Adapter Deployment Descriptor custom property configuration method.There is no default Export that supports inbound request from an Oracle Tuxedo application domain to a Java Application Server; this is true for all three configuration styles.

Factory-Based Configuration

Factory-based configuration is similar to Resource Adapter Deployment Descriptor-based configuration that they both utilize custom property to configure a Tuxedo JCA Adapter. The difference between the two methods is that Factory-Based Configuration configures connection factory custom properties.Factorybased configuration provides a better configuration support using a larger set of custom properties.

Factory-based configuration consists of two major parts. The first part is adapter-wise properties that need to be configured in the Deployment Descriptor file in the "`resourceadapter`" using custom property. The second part is factory-wise properties that can are configured differently for different Java Application Servers. For WebSphere, they are configured through the custom

properties page of the "J2C connection factory" For WebLogic they are configured in `weblogic-ra.xml`.

Normally, you configure the Tuxedo JCA Adapter after it has been installed. You configure Resource Adapter Deployment Descriptor, and then configure custom factory properties.

Properties In Resource Adapter Deployment Descriptor

There is a set of custom properties in the ResourceAdapter Deployment Descriptor that are supported by factory-base configuration. They are there because they are all adapter-wise properties.

These properties are based on standard simple property types: `config-property-name`, `config-property-type`, and `config-property-value`. These property names cannot be repeated. They are available in the Resource Adapter Deployment Descriptor `"resourceadapter"`.

The Resource Adapter Deployment Descriptor custom properties for factory-based configuration supports the following types of properties:

Adapter-Wise Properties

Resource-Related Properties

Adapter-Wise Properties

Adapter-wise properties are available to all connection factories.   There are a few adapter-wise properties available to factory-based configuration; however, they can be overridden by properties with the same name in the factory configuration. If these properties are not configured in a factory, the factory uses these adapter-wise properties configuration.

Table 20 lists these adapter-wise custom properties in the Resource Adapter Deployment Descriptor that are supported by factory-based configuration.

Table 20 Adapter-Wise Properties

Property	Type	Default Value	Description
`autoTran`	`Boolean`	`false`	Defines whether `AUTOTRAN` is allowed or not. It is used by a factory if factory-wise autoTran is not configured.
`appManagedLoc alTxTimeout`	`Integer`	`300 seconds`	Define the transaction timeout used by `AUTOTRAN` or client application managed local transaction. It is used by a factory if factory-wise `appManagedLocalTxTimeout` is not configured.
`throwFailureR eplyException`	`Boolean`	`true`	Configures whether a `com.oracle.tuxedo.adapter.Tuxed oReplyException` is thrown or not if a failure reply is received from Oracle Tuxedo. It is used by a factory if factory-wise `throwFailureReplyException` is not configured for that factory.

The following is the precedence order for `AUTOTRAN` transaction timeout in factory-based configuration:

1. factory-wise `appManagedLocalTxTimeout` property

2. adapter-wise `appManagedLocalTxTimeout` property

3. `com.oracle.tuxedo.adapter.AppManagedLocalTxTimeout` JVM property

4. default to 300 seconds

The adapter-wise "`appManagedLocalTxTimeout`" is configured in "`resourceadapter`" type in the Resource Adapter Deployment Descriptor (RADD), ra.xml, file as "`config-property`".

Listing 21 shows an example using `AUTOTRAN` with transaction timeout in the `ra.xml` file.

Listing 21   using AUTOTRAN with Transaction Timeout in ra.xml File Example

```
<resourceadapter>
```

```
<resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoFBCResourceAdapter<
/resourceadapter-class>

   <config-property>
     <config-property-name>autoTran</config-property-name>
     <config-property-type>java.lang.Boolean</config-property-type>
     <config-property-value>true</config-property-value>
   </config-property>
   <config-property>

     <config-property-name>appManagedLocalTxTimeout</config-property-name>
     <config-property-type>java.lang.Integer</config-property-type>
     <config-property-value>50</config-property-value>
   <config-property>

…
```

Resource-Related Properties

The Resources-related properties are available for every factory, and configured using Resource Adapter Deployment Descriptor.   They are configured in the "resourceadapter" type in the Deployment Descriptor. The only exception is Application Password that it is made available to each factory for flexibility and is not available in Resources Related properties.

All properties listed in Resource Related Properties Table 21 are optional.

Table 21 Resource Related Properties

Property	Type	Default Value	Description
fieldTable16C lasses	String	None	Field table 16 classes for FML. It is a comma-separated list.
fieldTable32C lasses	String	None	Field table 32 classes for FML32. It is a comma-separated list.
viewFile16Cla sses	String	None	VIEW table 16 classes for VIEW. It is a comma-separated list.

Table 21 Resource Related Properties

Property	Type	Default Value	Description
viewFile32Cla sses	String	None	VIEW table 32 classes for VIEW32. It is a comma-separated list.
tpusrFile	String	None	The full path name to the TPUSR file.
remoteMBEncod ing	String	None	The Multi-Bytes encoding used by an Oracle Tuxedo Application.
mBEncodingMap File	String	None	Full path name to the encoding map file.

Listing 22 shows a configuration example that describes two VIEW32 classes (view1 and view2), information in the Resource Adapter Deployment Descriptor for factory-based configuration.

Listing 22 Resource Adapter Deployment Descriptor Factory-Based Configuration

```
…

<resourceadapter>

<resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoFBCResourceAdapter<
/resourceadapter-class>

  <config-property>

   <config-property-name>viewFile32Classes<config-property-name>

   <config-property-type>java.lang.String</config-property-type>

   <config-property-value>tuxedo.view32.view1,tuxedo.view32.view2
</config-property-value>

  </config-property>

…
```

Connection Factory Properties

There are three property types that can be configured for a connection factory:

- adapter-wise property (also available as factory property).

- property unique to factory-based configuration.

- regular configuration property.

This section contains the following topics:

- Adapter-Wise Properties Also Available In Factory

- Connection Factory Name

- Application Password

- Local Access Point Related Properties

- RemoteAccess Point-Related Properties

- RemoteAccess Point-Related Properties

- Session Profile Related Properties

- Import-Related Property

- Session

- Export

Adapter-Wise Properties Also Available In Factory

A few adapter-wise properties can be specified for each factory configuration, and they override the adapter-wise properties for factories. If they are not specified in a factory, the factory uses the adapter-wise configuration for these properties.

Table 22 Factory-Wise Property Table

Property Name	Type	Default Value	Description
`autoTran`	Boolean	None	Defines whether `AUTOTRAN` is available for requests using connections created by this factory. If not configured then use adapter-wise `AUTOTRAN` configuration. If configured, whether `true` or `false`, will override adapter-wise `AUTOTRAN` configuration.
`appManagedLoc alTxTimeout`	Integer	None	Defines the transaction timeout used by `AUTOTRAN` or client application managed transaction. If not configured then use adapter-wise local TX timeout. It is measured in seconds.
`throwFailureR eplyException`	Boolean	None	Configures whether a `com.oracle.tuxedo.adapter.Tuxe doReplyException` is thrown or not if a failure reply received from Oracle Tuxedo. If it is not configured then adapter-wise setting is used.

The following is the precedence order for `AUTOTRAN` in a factory-based configuration.

1. factory `appManagedLocalTxTimeout` property

2. adapter-wise `appManagedLocalTxTimeout` property

3. `com.oracle.tuxedo.adapter.AppManagedLocalTxTimeout` JVM property

4. default to 300 seconds

Listing 23 shows a `weblogic-ra.xml` file example.

Listing 23   weblogic-ra.xml File

...

```
<outbound-resource-adapter>
```

```
   <connection-definition-group>

<connection-factory-interface>javax.resources.cci.ConnectionFactory</conne
ction-factory-interface>

    <connection-instance>

      <jndi-name>eis/TuxedoConnectionFactory1</jndi-name>

        <connection-properties>

          <properties>

            <property>

              <name>autoTran</name>

              <value>true</value>

            </property>

            <property>

              <name>appManagedLocalTxTimeout</name>

              <value>50</value>

            </property>

…
```

Connection Factory Name

A connection factory name can be specified by using connectionFactoryName property.
Although this property is optional, it is recommended for configuration if a transaction is possible
for service requests originated using a connection that is created by this connection factory. It is
also recommended if you want to use DMMIB to configure DM_REMOTE_DOMAINS in an Oracle
Tuxedo /Domain configuration dynamically.

Table 23 lists connection factory properties.

Table 23 Connection Factory Property Table

Property Name	Type	Default Value	Description
connectionFac toryName	String	None	Defines the name of this connection factory.

Listing 24 shows a `weblogic-ra.xml` file example.

Listing 24   Connection Factory weblogic_ra-xml File

```
…

<outbound-resource-adapter>

  <connection-definition-group>


<connection-factory-interface>javax.resources.cci.ConnectionFactory</conne
ction-factory-interface>

    <connection-instance>

      <jndi-name>eis/TuxedoConnectionFactory1</jndi-name>

        <connection-properties>

          <properties>

            <property>

               <name>autoTran</name>

               <value>true</value>

            </property>

         <property>

             <name>connectionFactoryName</name>

            <value>TuxedoConnectionFactory1</value>

          </property>

    …
```

If this property is configured and `default LocalAccessPoint` is configured then a file with the name "`.lapid.<connectionFactoryName>`" will be created in the current working directory which will contains the LocalAccessPoint Id generated dynamically. For instance, using the above as an example, a file with the name "`.lapid.TuxedoConnectionFactory1`" will be created.

Application Password

The application password of the Resources, which is not supported in RADD-based configuration, can be configured in a factory-based configuration as factory-wise property. This property is available per factory, this is to facilitate the ability of different factory to have different Application Password. There is no equivalent in a RADD-based configuration.

The `applicationPassword` property for factory-based configuration as shown in Table 24, can be in either clear text or cipher text. To configure it using cipher text, you must use the output of `com.oracle.tuxedo.tools.EncryptPassword`. The following is the sample output:

```
c:\tuxedo\JCA\adapter> java -classpath %classpath%
com.oracle.tuxedo.tools.EncryptPassword mypassword foo.key
Encrypted Password: {Salted-AES}WBGk6LjHuI515pwXPTfaOQ==
```

For WebSphere 7.0 and above, there is no need to use this tool to encrypt password; WebSphere encrypts it for you.

Table 24 Application Password Property

Property Name	Type	Default Value	Description
applicationPa ssword	String	None	Oracle Tuxedo Application Password in either clear text, or cipher text generated using the `com.oracle.tuxedo.tools.Encrypt Password` tool.

Local Access Point Related Properties

Table 25 is the table for `LocalAccessPoint` related properties for factory-based configuration.

Table 25 Properties of LocalAccessPoint

Property Name	Type	Default Value	Description
localAccessPointSpec	String	None	Used by default as connection principal name. It has to be globally unique. `//<network address>:<port>/domainId`
mutualAuthenticationRequired	Boolean	false	Indicates whether mutual authentication is required when connecting to remote an Oracle Tuxedo GWTDOMAIN gateway. This for SSL.
identityKeyStoreFileName	String	None	The full path name of the identity key store file name. This is for SSL.
identifyKeyStorePassPhrase	String	None	The password for the identity key store in either clear text or cipher text. This is for SSL.
privateKeyAlias	String	None	The alias in the identity key store to be used to retrieve private key. This is for SSL.
privateKeyPassPhrase	String	None	The password, in either clear text or cipher text, used for decrypt the private key in the identity key store. This is for SSL.
trustKeyStoreFileName	String	None	The full path name of the trusted key store file name. This is for SSL.
trustKeyStorePassPhrase	String	None	The password, in either clear text or cipher text, to be used when retrieving certificate from trust key store. This is for SSL.

There are seven SSL-related properties. Six of them are related to Key/Certificate store and must be configured if SSL is required; the only one that is optional for using SSL is `mutalAuthenticationRequired`.

By default `"mutualAuthenticationRequire"` is `false`. If anyone of the six required properties is missing, SSL is ignored. Toni depends on session profile information plus session negotiation with remote Oracle Tuxedo GWTDOMAIN gateway and uses LLE.

Specifying `localAccessPointSpec` property is optional. If not specified, `default` `LocalAccessPoint` is used. When `default LocalAccessPoint` is used for this factory, it is recommended to also configure `connectionFactoryName`.

Listing 25 shows a `weblogic-ra.xml` file example.

Listing 25   weblogic-ra.xml Usage Example

```
…
<outbound-resource-adapter>

  <connection-definition-group>

<connection-factory-interface>javax.resources.cci.ConnectionFactory</conne
ction-factory-interface>

    <connection-instance>

      <jndi-name>eis/TuxedoConnectionFactory1</jndi-name>=

        <connection-properties>

          <properties>

            <property>

              <name>localAccessPointSpec</name>

              <value>//localhost:123456/domainId=JDOM</value>

            </property>
…
```

RemoteAccess Point-Related Properties

The `RemoteAccessPoint` is represented by both networking address and per `RemoteAccessPoint` Access Control related information. The most important property is `remoteAccessPointSpec`. It is a comma-separated list; a comma separates each `RemoteAccessPoint`. Table 26 lists the RemoteAccessPoint related properties that are available in factory-based configuration.

Table 26 RemoteAccessPoint Property Table

Property Name	Type	Value Range	Description
remoteAccessP ointSpec	String	No default value.	This property contains both NetworkAddress and AccessPointId plus the name attribute. This is a comma separated list of RemoteAccessPoint. The domainId in each entry is used to replace AccessPointId. This domainId is used by default as connection principal name that is used to identify a remote access point when attempting to establish a session with remote Tuxedo access point. If not specified then it is an error, and it has to be globally unique.
			The NetworkAddress also become part of the specification. The NetworkAddress contains both host network address and port number of the remote Tuxedo access point. Specify the TCP/.IP address in the format of //hostname:port or //#.#.#.#:port.
rapAllowAnony mous	Boolean	{true, false}	Indicates whether the remote Tuxedo access point allow anonymous access or not. The default value is "false".
rapDefaultApp licationKey	String	Any valid Tuxedo application key.	The default application key value for this remote Tuxedo access point. If not specified then "-1" is assumed.
rapApplicatio nKeyClass	String	No default value.	The fully qualified class name of the custom application key generator. If not specified then the default application key generator will be used.
rapApplicatio nKeyClassPara m	String	No default value.	The parameter string passed to the custom the application key generator when the class is initialized at runtime.

Each RemoteAccessPoint is represented in a specific format. In order to make the factory usable a "remoteAccessPointSpec" must be configured.

shows a RemoteAccessPoint `weblogic-ra.xml` file example.

Listing 26   RemoteAccessPoint weblogic-ra.xml File

```
…

<outbound-resource-adapter>

  <connection-definition-group>


<connection-factory-interface>javax.resources.cci.ConnectionFactory</conne
ction-factory-interface>

    <connection-instance>

      <jndi-name>eis/TuxedoConnectionFactory1</jndi-name>

        <connection-properties>

          <properties>

            <property>

              <name>autoTran</name>

              <value>true</value>

            </property>

            <property>

              <name>localAccessPointSpec</name>

              <value>//localhost:123456/domainId=JDOM</value>

            </property>

           <property>

            <name>remoteAccessPointSpec</name>


<value>//bluestar:11023/domainId=guinevre,//orion:37654/domainId=galahad<v
alue>

           </property>

…
```

There can be only one `remoteAccessPointSpec` specified for each factory. If `rapApplicationKeyClass` and `rapApplicationKeyClassParam` are specified, they are used for identity propagation for both `guinevre` and `galahad`.

The RemoteAccessPointSpec

The `RemoteAccessPointSpec` property has been enhanced to be able to configure more RemoteAccessPoint and Session related attributes. Comma is used to separate these attributes. Each attribute is a name and value pair. The following is the list of attributes supported.

- `domainId` - The remote access point Id. It must be specified.

- `lPasswd1` - Local password of password pair 1, can be in either clear text or cipher text.

- `lPasswd2` - Local password of password pair 2, can be in either clear text or cipher text.

- `rPasswd1` - Remote password of password pair 1, can be in either clear text or cipher text.

- `rPasswd2` - Remote password of password pair 2, can be in either clear text or cipher text.

At least one password pair must be valid if session authentication is "`DM_PW`". If both password pair 1 and password pair 2 are valid then password pair one will be used to encrypt session authentication information.

The `lPasswd1`, `lPasswd2`, `rPasswd1`, and `rPasswd2` attributes for factory-based configuration can be in either clear text or cipher text. To configure using cipher text, you must use the output of `com.oracle.tuxedo.tools.EncryptPassword`. The following is the sample output:

```
c:\tuxedo\JCA\adapter> java -classpath %classpath%
com.oracle.tuxedo.tools.EncryptPassword mypassword foo.key
Encrypted Password: {Salted-AES}WBGk6LjHuI515pwXPTfaOQ==
```

Listing 27 shows a `RemoteAccessPointSpec` property in the `weblogic-ra.xml` file example.

Listing 27   RemoteAccessPointSpec Property weblogic-ra.xml File

```
…
<outbound-resource-adapter>

  <connection-definition-group>
```

```
<connection-factory-interface>javax.resources.cci.ConnectionFactory</conne
ction-factory-interface>

    <connection-instance>

      <jndi-name>eis/TuxedoConnectionFactory</jndi-name>

      <connection-properties>

        <properties>

          <property>

            <name>spSecurity</name>

            <value>DM_PW</name>

          </property>

          <property>

            <name>remoteAccessPointSpec</name>

<value>//localhost:123456/domainId=TUX_ID,lPasswd1=weblogic,rPassword=tuxe
do</value>

            </property>
…
```

Session Profile Related Properties

Table 27 lists the default value of the default `SessionProfile` for factory-based configuration.

Table 27 SessionProfile Property Table

Property Name	Type	Value Range	Description
spBlockTime	Integer	0..2147483647	The maximum number of milliseconds allowed for a blocking outbound request using this profile. The default value is 60 seconds, which is 60000 milliseconds. The InteractionSpec.setExecutionTime() time method can override this block timeout.
spSecurity	String	{APP_PW,DM_PW,NONE}	If not configured then GWTDOMAIN session authentication is not required. When DM_PW session authentication security type is configured then at least one password pair must be configured for every remoteAccessPoint. If APP_PW session authentication security type is configured, you must configure applicationPassword for the factory.
spInteroperate	Boolean	{true, false}	Specifies whether this session is allowed to interoperate with a remote Tuxedo 6.5 release GWTDOMAIN gateway or not. The default value is "false".
Connection Policy	Not available		Behaves like ON_STARTUP.
spCredentialPolicy	String	{LOCAL. GLOBAL}	The user credential propagation policy. When its value s "LOCAL", there is no propagation. Default value is "LOCAL".
spRetryInterval	Long	0..2147483647	The number of seconds that this session waits between automatic connection establishment attempt. Set this element value only when ConnectionPolicy is "ON_STARTUP". Default value is 60. The value 0 disables the connection retry mechanism.

Table 27 SessionProfile Property Table

Property Name	Type	Value Range	Description
spMaxRetries	Long	0..9223372063 85775807	The maximum number of times that this session tries to establish a session connection to remote Oracle Tuxedo access points. Set this element only when `ConnectionPolicy` is `"ON_STARTUP"`. Default value is `9223372036385775807`.
spCompression Limit	Integer	0..2147483647	The compression threshold this session uses when sending data to a remote Oracle Tuxedo Access Point. Application buffers larger than this size are compressed. The default value is `2147483647`.
spMinEncryptB its	String	{"0","40","56 ","128","256" }	The minimum encryption key length (in bits), this session uses when establishing a session connection. A value of `0` indicates encryption is optional. Value `"256"` is for SSL. Default value is `"0"`.
spMaxEncryptB its	String	{"0","40","56 ","128","256" }	The maximum encryption key length (in bits), this sessions uses when establish a session connection. A value of `0` indicates no encryption is used. The default value is `"128"`.

Table 27 SessionProfile Property Table

Property Name	Type	Value Range	Description
spKeepAlive	Long	{0..2147483647}	Specifies whether this GWTDOMAIN session is configured with Application-Level Keep Alive, and its maximum idle time before wait timer start ticking. Default value is "0", and it means application level keep alive is disabled. The measurement is in milliseconds.
			When connection is busy, there is no need to send a special keep alive message to remote gateway; however, when there is spKeepAlive number of milliseconds, rounded up to second, without activities over the connection then a special keep alive message is sent and a timer of spKeepAliveWait (rounded up to second) will be started.
			If no acknowledgement received during this "wait" time the connection is declared dead, connection is closed, and the session is terminated.
spKeepAliveWait	Long	{0..2147483647}	Tells whether this session requires the acknowledgement of Application Level Keep Alive or not, and how long it will wait without receiving acknowledgement before declare the connection is inaccessible. Default value is 10 seconds.
			If the value specified is '0' then there will be no checking the acknowledgement from RemoteAccessPoint; this can prevent the session connection being closed by KeepAlive feature. The measurement is in milliseconds.

Listing 28 shows a SessionProfile Property `weblogic-ra.xml` file example.

Listing 28   SessionProfile Property weblogic-ra.xml File

```
…

<connection-instance>

…

  <connection-properties>

…

    <properties>

      <property>

        <name>spBlockTime</name>

        <value>120000</value>

      </property>

…

</connection-instance>
```

Import-Related Property

Factory- based configuration can utilize the default Import; however, it will also provide ability to restrict what can be accessed from adapter to remote Tuxedo Application Domain in a uniformly way. A single `"impResourceName"` property can be specified for each factory and it contains a comma-separated list of remote Oracle Tuxedo service/resources the Tuxedo JCA Adapter client is allowed to access.

The `LoadBalancing` algorithm cannot be specified and it will always use RoundRobin. Your service requests are load balanced among all the `RemoteAccessPoints` of that particular connection factory.

Table 28 lists the new property related to Import in a factory-based configuration.

Table 28 Import Related Property weblogic-ra.xml File

Property Name	Type	Value Range	Description
impResourceNa me	String	Any valid Tuxedo resource name in a comma-separat ed list.	The valid Oracle Tuxedo service name or queue name. There is no name translation done for the wire protocol. This is essentially the same as the `RemoteName` attribute of an Import in the `dmconfig` file.

Listing 29 shows a `weblogic-ra.xml` file example.

Listing 29   Weblogic-ra.xml Usage Example

```
…

<connection-instance>

…

  <connection-properties>

    <properties>

      <property>

        <name>impResourceName</name>

        <value>TOUPPER,ECHO</value>

      </property>

…

</connection-instance>
```

Session

A session connects a Tuxedo JCA Adapter `LocalAccessPoint` to a remote Oracle Tuxedo GWTDOMAIN gateway. In a factory-based configuration, there is no need to explicitly specify

a session so there is no "Session" related property available. All the sessions available in a factory will be default Session.

The Tuxedo JCA Adapter creates a session for every possible LocalAccessPoint and RemoteAccessPoint combinations for a connection factory. Since there can only be one LocalAccessPoint per connection factory configuration, there can be at most 1xN number of sessions possible (where the 'N' is the number of the RemoteAccessPoint). Listing 30 shows an example using Session.

Listing 30   Session weblogic-ra.xml.

```
<property>

  <name>localAccessPointSpec</name>

  <value> //localhost:12345/domainId=JDOM</value>

</property>

<property>

  <name>remoteAccessPointSpec</name>


<value>//localhost:13456/domainId=TDOM1,//blues:23457/domainId=TDOM2</valu
e>

</property>
```

Then the following sessions are possible.

```
(JDOM,TDOM1)
(JDOM,TDOM2)
```

Export

There is no "Export" related property available for factory-based configuration.

How To Configure Factory-Based Configuration

- JBOSS Server
- WebLogic Server

- WebSphere Server

JBOSS Server

The Tuxedo JCA Adapter factory-based configuration is shown in Listing 31. The file name can be anything you want, take `TJA-ds.xml` for example, this file should be copied to the `$JBOSS_HOME/server/$BOOTMODE/deploy` directory.

Listing 31   JBOSS Server Sample Configuration

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

<connection-factories>

    <tx-connection-factory>

        <jndi-name>eis/TuxedoConnectionFactory</jndi-name>

        <use-java-context>false</use-java-context>

        <rar-name>com.oracle.tuxedo.TuxedoAdapter.rar</rar-name>


<connection-definition>javax.resource.cci.ConnectionFactory</connection-de
fini

tion>

        <max-pool-size>50</max-pool-size>

        <config-property name="autoTran"

type="java.lang.Boolean">true</config-property>

        <config-property name="connectionFactoryName"

type="java.lang.String">Factory1</config-property>

        <config-property name="localAccessPointSpec"

type="java.lang.String">//bej301151:2497/domainId=JDOM</config-property>

        <config-property name="remoteAccessPointSpec"

type="java.lang.String">//bej301151:3138/domainId=TDOM1</config-property>

        <config-property name="spBlockTime"

type="java.lang.Integer">20000</config-property>
```

```
        <config-property name="spInteroperate"

type="java.lang.Boolean">false</config-property>

        <config-property name="spCredentialPolicy"

type="java.lang.String">Local</config-property>

        <config-property name="spRetryInterval"

type="java.lang.Long">60</config-property>

        <config-property name="spMaxRetries"

type="java.lang.Long">1000</config-property>

        <config-property name="spCompressionLimit"

type="java.lang.Integer">1000000</config-property>

        <config-property name="spMinEncryptBits"

type="java.lang.String">56</config-property>

        <config-property name="spMaxEncryptBits"

type="java.lang.String">128</config-property>

        <config-property name="spKeepAlive"

type="java.lang.String">0</config-property>

        <config-property name="spKeepAliveWait"

type="java.lang.Long">200000</config-property>

        <config-property name="impResourceName"

type="java.lang.String">UPDATE_DB1,UPDATE_DB2,CLEAN_DB1,CLEAN_DB2,COUNT_DB
1,CO

UNT_DB2,UPDATE_FAIL_DB1,TIMEOUT_DB1,FAILSVC_DB1,FAILSVC_DB2,UPDATE_FAIL_DB
2</c

onfig-property>

    </tx-connection-factory>

</connection-factories>
```

WebLogic Server

Tuxedo JCA Adapter factory-based configuration is done through the WebLogic `weblogic-ra.xml` file. Examples of the configuration can be found in previous sections; Listing 32 lists a complete sample configuration.

Listing 32   WebLogic Server Sample Configuration

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

<weblogic-connector

 xmlns="http://www.bea.com/ns/weblogic/90">

  <jndi-name>eis/TuxedoConnector</jndi-name>

  <enable-access-outside-app>true</enable-access-outside-app>

  <enable-global-access-to-classes>true</enable-global-access-to-classes>

  <outbound-resource-adapter>

    <connection-definition-group>

<connection-factory-interface>javax.resource.cci.ConnectionFactory</connec
tion-factory-interface>

      <connection-instance>

        <jndi-name>eis/TuxedoConnectionFactory1</jndi-name>

        <connection-properties>

          <properties>

            <property>

              <name>remoteAccessPointSpec</name>

              <value>//localhost:12478/domainId=TDOM1_ID</value>

            </property>

          </properties>

        </connection-properties>

      </connection-instance>
```

```
<connection-instance>

  <jndi-name>eis/TuxedoConnectionFactory2</jndi-name>

  <connection-properties>

    <properties>

      <property>

        <name>spSecurity</name>

        <value>APP_PW</value>

      </property>

      <property>

        <name>applicationPassword</name>

        <value>{Salted-AES}hHAsW13whgqTobG1t9Q92Q==</value>

      </property>

      <property>

        <name>remoteAccessPointSpec</name>

        <value>//localhost:12488/domainId=TDOM2_ID</value>

      </property>

    </properties>

  </connection-properties>

</connection-instance>

<connection-instance>

  <jndi-name>eis/TuxedoConnectionFactory3</jndi-name>

  <connection-properties>

    <properties>

      <property>

        <name>spSecurity</name>

        <value>DM_PW</value>

      </property>
```

```
               <property>

                 <name>localAccessPointSpec</name>

                 <value>//localhost:10801/domainId=JDOM_ID</value>

               </property>

               <property>

                 <name>remoteAccessPointSpec</name>

                 <value>//localhost:12498/domainId=TDOM3_ID,

                 lPasswd1={Salted-AES}xNgOdUuXB7Z49D0cssluxA==,

                 rPasswd1={Salted-AES}hAIzbPI+YyaeuHX0A9Umqg==</value>

               </property>

            </properties>

          </connection-properties>

        </connection-instance>

     </connection-definition-group>

  </outbound-resource-adapter>

</weblogic-connector>
```

WebSphere Server

Normally, you configure the Tuxedo JCA Adapter after it has been installed.You can configure RADD-based adapter-wise property first, and then configure factory property.

Configure Deployment Descriptor Property

The following is the procedure to configure adapter-wise property in the RADD from the WebSphere Console.

1. Install Tuxedo JCA Adapter from the console.

   Resources ' `Resource Adapters`' Resource adapters

2. Click on the "name" column of the installed Tuxedo JCA Adapter.

   Assume you use the name "Tuxedo JCA Adapter"

3. From Resources ' `Resource Adapters` ' Resource adapter ' Tuxedo JCA Adapter.

   Click on "Custom properties" on the right hand side under "Additional Properties".

4. From Tuxedo JCA Adapter ' Custom properties

   Click on "Preference" on top of custom property table. The console expands to add an input field "Maximum rows", change the default value to "`60`", and then click on "Apply" button right below the input field box and check box.

   You should be able to see all the configurable adapter-wise properties.

5. Click on the desired property name under the "Name" column.

   For instance: "`remoteAccessPointSpec`"

6. From Tuxedo JCA Adapter ' Custom properties ' `remoteAccessPointSpec`

   Change or add the desired `remoteAccessPointSpec` value to the "Value" field, then click on the "Apply" button and the click on "Save".

7. Use step #5 to step #6 to modify another property.

If you configure `applicationPassword` property for WebSphere 7.0, you should not encrypt the password using `com.oracle.tuxedo.tools.EncryptPassword` tool because WebSphere 7.0 will encrypt the password.

Configure Factory Property

To configure factory-based configuration for WebSphere, do the following steps:

1. Install Tuxedo JCA Adapter from console if it has not yet been done. (Resources ' Resource Adapters ' Resource adapters).

2. Click on the "name" column of Tuxedo JCA Adapter.

   Resources ' Resource Adapters ' Resource adapters

   Assumes you give name "Tuxedo JCA Adapter".

3. From Resources ' Resource Adapters ' Resource adapters ' Tuxedo JCA Adapter.

   Click on "J2C connection factories" on the right hand side under "Additional Properties".

4. From Resources ' Resource Adapters ' Resource adapters ' Tuxedo JCA Adapter ' J2C connection factories.

   Click on button "New".

5. From J2C connection factories ' New,

enter the connection factory name in the "Name" field, and then enter the unique JNDI name in the "JNDI name" field, then click on "Apply" button at bottom of the page.

Assume you entered "`factory1`" in the "Name" field.

6. From J2C connection factories ' factory1,

   click on the "Save" button. This will take you back to "J2C connection factories" page.

7. From Tuxedo JCA Adapter ' J2C connection factories,

   click on the "`factory1`" in the "Name" column, and then it will take you to page "factory1".

8. From Tuxedo JCA Adapter ' J2C connection factories ' factory1,

   click on the "Custom properties" in the right hand side under "Additional Properties".

9. From Tuxedo JCA Adapter ' J2C connection factories ' `factory1` ' Custom properties,

   click on the property name in the "Name" column and then it will take you to property page. Assume you selected "`localAccessPointSpec`".

10. From J2C connection factories ' factory1 ' Custom properties ' `localAccessPointSpec`,

    enter the desired value in the "Value" field, then click on "Apply" button, and then click on "Save". This will take you back to "Custom properties" page.

11. Repeat steps #9 and #10 for each property that need to be changed.

Oracle Tuxedo JCA Adapter Deployment

Deployment of the Resource Adapter is Application Server dependent. This is usually achieved by:

- Dropping the `.rar` file in a generic auto-deployment location.

- Explicitly deploying the `.rar` file (or the directory containing its exploded version) via a console function.

- Using an application supported scripting tool to install and deploy. For example `wsadmin` in WebSphere or `wldeployer` in WebLogic.

- Using system class path to load VIEW and FML classes, so it is also necessary to configure system class path.

● Using other web or application deployment mechanisms specified by the Application Server

Deploying the Tuxedo JCA Adapter involves choosing a deployment mode, configuring the resource adapter descriptor, repackaging the adapter and deploying it on a JCA 1.5/1.6 compliant JEE application server. In most cases, you only need to modify the `ra.xml` file (in the `META-INF` directory) to get the adapter up and running.

The section addresses the following topics:

● Oracle JCA Adapter Deployment Tasks

● Repackaging the Oracle Tuxedo JCA Adapter

● Changing the Connector Connection Pool Size.

Notes: In most cases, you only need to modify the `ra.xml` file (in the `META-INF` directory) to get the adapter up and running.

The resource archive configuration and repackaging procedure is the same regardless the type of the targeted application; however, the deployment of the Tuxedo JCA Adapter is different from application server to application server.

For more information, see your target application server documentation.

Oracle JCA Adapter Deployment Tasks

The following tasks are required in order to deploy the Oracle JCA Adapter:

1. Unjar the `com.oracle.tuxedo.TuxedoAdapter.rar` file into a directory.

2. Decide whether to use Client-Side only or full blown server and client.

    a. Choose to use Client-Side only

      ● remove `META-INF/ra.xml`

      ● remove `META-INF/server.ra.xml`

      ● remove `META-INF/sample.weblogic-ra.xml`

      ● rename `META-INF/client-side.ra.xml` to `META-INF/ra.xml`

      ● modify `META-INF/ra.xml` by adding properties for the /Domain configuration.

    b. Choose to use full blown server and client operations

      ● remove `META-INF/ra.xml`

- remove `META-INF/client-side.ra.xml`

- remove `META-INF/sample.weblogic-ra.xml`

- rename `META-INF/server.ra.xml` to `META-INF/ra.xml`

- modify `ra.xml` file with the desired /Domain configuration properties, `'dmconfig'` property must be specified

- modify the `dmconfig.xml` with the desired and correct /Domain configuration

- rename the `dmconfig.xml` file to whichever name `'dmconfig'` property specified.

- if you choose the `'dmconfig'` file not to be treated as part of resource archive then move it to desired directory.

  c. Choose to use Factory-based Configuration

- remove `META-INF/client-side.ra.xml`

- remove `META-INF/server.ra.xml`

- remove `META-INF/weblogic-ra.xml`

- rename `META-INF/sample.weblogic-ra.xml` to `META-INF/weblogic-ra.xml` if configuring it for WebLogic server.

- For WebLogic server user modifies the `META-INF/weblogic-ra.xml` with the desired configuration. For WebSphere server user use the steps described in previous chapter to configure it from console.

3. `.jar` the working directory to create Resource Adapter Archive

4. Deploy the Resource Adapter Archive with the preferred application server method you want to use.

    Some application servers may need to configure connection pool information through the console. Some application servers may need to 'activate' the adapter

Configuring Oracle Tuxedo JCA Adapter Deployment

Check the configured adapter class for Tuxedo JCA Adapter is one of the following:
`TuxedoResourceAdapter`, or `TuxedoClientSideResourceAdapter`, or
`TuxedoFBCResourceAdapter`.

TuxedoResourceAdapter Class

If `TuxedoResourceAdapter` is used, then the `dmconfig` property must be configured. If the configured `dmconfig` property contains only file name without any path information, the configuration is loaded as a resource as shown in Listing 33.

If it fails to load from the resource archive, it is treated as a file located in the current working directory. If the file fails to open, the resource adapter will not start and a `ResourceAdapterException` is thrown.

Listing 33   dmconfig Custom Property to Loaded dmconfig as a Resource

```
<config-property>
  <config-property-name>dmconfig</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   <config-property-value>dmconfig.xml</config-property-value>
</config-property>
```

If the configured `dmconfig` property contains path information, it is treated and loaded as file as shown in Listing 34. If the file fails to open, the resource adapter will not start and a `ResourceAdapterException` is thrown.

Listing 34   dmconfig Property Loaded as a File

```
…
<config-property>
   <config-property-name>dmconfig</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   <config-property-value>/user/dilbert/tja/dmconfig.xml
   </config-property-value>
</config-property>
…
```

If the `dmconfig` file does not have any `LocalAccessPoint` configured, it creates a single default `LocalAccessPoint`. This default `LocalAccessPoint` can only have a session with

RemoteAccessPoint using the default SessionProfile; it can only initiate outbound connection.

The Adapter creates a default SessionProfile using all the default values (except for the ConnectionPolicy which is always ON_STARTUP for the default SessionProfile). If SessionProfile is configured in the dmconfig file, it is constructed in addition to the default SessionProfile.

If the dmconfig file does not have any Session configured, the adapter creates a default Session between each LocalAccessPoint and each RemoteAccessPoint configured using default SessionProfile.

The resourceadapter-class element in the resource descriptor ra.xml file should contain the com.oracle.tuxedo.adapter.TuxedoResourceAdapter fully qualified class name as its value as shown in Listing 35.

Listing 35   resourceadapter-class Element - com.oracle.tuxedo.adapter.TuxedoResourceAdapter

```
…
<resourceadapter>
  <resourceadapter-class>com.oracle.tuxedo.adapter.
   TuxedoResourceAdapter</resourceadapter-class>

…
```

Note: The new "client-side" mode properties are not available to this class-based resource adapter. If you configure these properties in the Resource Adapter Deployment Descriptor file, the behavior is application server dependent.

TuxedoClientSideResourceAdapter Class

If TuxedoClientSideResourceAdapter is configured, dmconfig configuration is ignored. When this class of resource adapter is configured it assumes all configuration information is in the resource adapter Java Bean provided by the application server JCA container.

If no localAccessPointSpec property is configured, a default LocalAccessPoint is created for the Resource Adapter Deployment Descriptor file-based configuration.

If `remoteAccessPointSpec` property is configured, it is used to construct `RemoteAccessPoint`. If there is no `remoteAccessPointSpec` property configured, the configuration cannot be used and a warning message is logged in the adapter log file.

A default `SessionProfile` is created using information from properties related to Session Profile. If no session profile related properties are configured, the default `SessionProfile` is constructed using only the default values. It creates sessions from the `LocalAccessPoint` to every `RemoteAccessPoint` using the default `SessionProfile`.

The `resourceadapter-class` element in the resource descriptor `ra.xml` file, should contain the `com.oracle.tuxedo.adapter.TuxedoClientSideResourceAdapter` fully qualified class name as its value as shown in Listing 36.

**Listing 36   resourceadapter-class Element -
com.oracle.tuxedo.adapter.TuxedoClientSideResourceAdapter**

```
…
<resourceadapter>
  <resourceadapter-class>com.oracle.tuxedo.adapter.
   TuxedoClientSideResourceAdapter</resourceadapter-class>
…
```

TuxedoFBCResourceAdapter Class

If `TuxedoFBCResourceAdapter` is configured, `dmconfig` configuration and Resource Adapter Deployment Descriptor specific configuration properties are ignored. When this class of resource adapter is configured it assumes all the configuration information is in the resource adapter Java Bean provided by the application server JCA container and the Managed Connection Factory Java Bean that is also provided by the application server JCA container.

If no `localAccessPointSpec` property is configured for a factory, a default `LocaAccessPoint` is created for that factory. A `remoteAccessPointSpec` must be configured for each factory, and they will be used to construct `RemoteAccessPoint`.

A default `SessionProfile` is created for each factory using information from the properties related to `SessionProfile` of that factory. If no session profile related properties are configured for a factory, the factory uses the adapter-wise default `SessionProfile`.

The `resourceadapter-class` element in the resource deployment descriptor `ra.xml` file, should contain the `com.oracle.tuxedo.adapter.TuxedoFBCResourceAdapter` fully qualified class name as its value as shown in Listing 37.

Listing 37   Resource Deployment Descriptor ra.xml File resourceadapter-class Element

```
...
<resourceadapter>
   <resourceadapter-class>com.oracle.tuxedo.adapter.
    TuxedoFBCResourceAdapter</resourceadapter-class>
...
```

Resource Adapter Deployment Descriptor Properties

There is a set of Resource Adapter Deployment Descriptor custom properties that are available for all three styles configuration: Resource Adapter Deployment Descriptor configuration, factory-based configuration, and `dmconfig`-based configuration. The only exception is the "dmconfig" custom property that is only available using `dmconfig`-based configuration. If you specified this property while using a Resource Adapter Deployment Descriptor-based configuration, this property is ignored.

Customizing Properties

Some of the properties in `ra.xml` file should not be changed as they pertain to the Tuxedo JCA Adapter internally or its descriptive information; however, there are other properties you must modify to customize the operation and application.

config-property

The `config-property` element is generally used to define Tuxedo JCA Adapter custom properties in the standard JCA deployment descriptor `META-INF/ra.xml` file. Table 29 lists the properties that are used to customize the Tuxedo JCA Adapter.

You must specify the Tuxedo JCA Adapter configuration file using the `dmconfig` property in the resource deployment descriptor `META-INF/ra.xml` file. For more information, see Configuration File Examples in the Tuxedo JCA Adapter Programming Guide.

Table 29 Customization Properties

Property	Type	Initial value (or default if not specified)	Description
traceLevel	java.lang.St ring	0	Level of debug tracing. For more information, see Transaction Support.
xaAffinity	java.lang.St ring	true	Turn on transaction-specific routing, for enhancing transaction performance.
keyFileName	java.lang.St ring	c:\tuxedo\ke yfile	For dmconfig-based configuration only, this is the full path name to the key file. This file contains key used to encrypt all the passwords in the "dmconfig" file.
dmconfig	java.lang.St ring	C:\tuxedo\co nfig.xml	For dmconfig-based configuration only, this is the full path name to the Tuxedo JCA Adapter configuration file. **Note:** If it does not contain path information then it will be treated as resource
appManagedLo calTxTimeout	java.lang.In teger	default value 300 seconds	Transaction timeout value of the local transaction managed by the Tuxedo JCA Adapter.
throwFailure ReplyExcepti on	java.lang.Bo olean	default value true	Customize CCI execution interface to decide whether to throw exception or failure reply data in the output data record in case an Oracle Tuxedo service returns failure reply.
autoTran	java.lang.Bo olean	no default value	Enables/disables AUTOTRAN. By default there is no AUTOTRAN.

Trace Level Support

Table 30 lists the trace-level control values.

Table 30 Trace Level Control Values

Values	Components Traced	Description
20000	GWT_IO	Gateway input and output, including the ATMI verbs.
25000	GWT_EX	More Gateway information.
50000	JATMI_IO	JATMI input and output, including low-level JATMI calls.
55000	JATMI_EX	More JATMI information.
100000	All components	Information on all Tuxedo JCA Adapter components.

Note: Tracing is not executed for trace levels less that 20,000.

Listing 38 shows an example deployment descriptor file using the customization properties.

Listing 38   Customized Deployment Descriptor File Example

```
<config-property>
  <config-property-name>traceLevel</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>0</config-property-value>
</config-property>
<config-property>
  <config-property-name>xaAffinity</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>true</config-property-value>
</config-property>
<config-property>
  <config-property-name>keyFileName</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
<config-property-value>foo.key</config-property-value>
</config-property>
<config-property>
  <config-property-name>dmconfig</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
```

```
<config-property-value>dmconfig.xml</config-property-value>
</config-property>
<config-property>
  <config-property-name>throwFailureReplyException</config-property-name>
  <config-property-type>java.lang.Boolean</config-property-type>
<config-property-value>true</config-property-value>
</config-property-value>
</config-property>
<config-property>
  <config-property-name>appManagedLocalTxTimeout</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>
  <config-property-value>15</config-property-value>
</config-property>
```

Repackaging the Oracle Tuxedo JCA Adapter

Most application servers allow the resource adapter to be deployed in un-archived form; however, it is best to repackage the adapter after modifying the resource deployment descriptor before it is deployed.

Repackaging requires converting the Tuxedo JCA Adapter into a resource archive using the `"jar"` command that comes with the JDK. The resource archive has the `".rar"` extension.

For example, use the following command from the root directory of resource archive:
```
jar -cvf ../com.oracle.tuxedo.TuxedoAdapter.rar *
```

Changing the Connector Connection Pool Size

The application server has a default connection pool size. For most applications, the default connection pool size is large enough; however, in some situations the default connection pool size may not be enough. For example, the Oracle WebLogic Server has a default connection pool size of 10 which means it can support a maximum of 10 concurrent JCA clients using the same adapter to access remote Enterprise Information Systems (EIS). If an application wants to support more than 10 concurrent clients using the Tuxedo JCA Adapter, the application must expand the connection pool size.

For example, to change the connection pool size from default value to 20 in an Oracle WebLogic Server installation, you can modify the `weblogic-ra.xml` file in the `META-INF` directory as shown in Listing 39.

Listing 39   Change Oracle WebLogic Connector Connection Pool Size

```
<outbound-resource-adapter>
  <default-connection-properties>
    <pool-params>
      <initial-capacity>15</initial-capacity>
      <max-capacity>20</max-capacity>
    </pool-params>
  </default-connection-properties>
  <connction-definition-group>
  ...
  </connection-definition-group>
</outbound-resource-adapter>
```

Note: Different application servers have different connection pool size configuration methods. Some application servers use the custom deployment descriptor to configure connection pool size (such as Oracle WebLogic Server). Other application servers use console configuration (such as IBM WebSphere).

For more information, see your application server documentation.

Oracle Tuxedo JCA Adapter Management

After deploying the Tuxedo JCA Adapter, you may want to expand or change the application. To do so, you must modify the adapter configuration. Modifying the Tuxedo JCA Adapter configuration involves using a text editor or XML editor to alter the contents of the configured adapter configuration.

For Oracle WebLogic Server and JBOSS application server environments, you must do the following steps:

1. Modify the configuration file.

2. Stop the adapter.

3. Restart the adapter with the new configuration file.

Note: For IBM WebSphere application servers, you must stop the application server and then restart it.

Runtime Configuration Updating

The Tuxedo JCA Adapter does not support runtime dynamic configuration modification without stopping and restarting for the configuration changes to take effect.

Oracle Tuxedo JCA Adapter Fail Over Capability

Tuxedo JCA Adapter supports both link level fail over and service level fail over. The link-level fail over gives user ability to specify different Access Point network addresses. This is applicable to both Local Access Point and Remote Access Point.

In `dmconfig` file configuration, you can specify multiple `NetworkAddress` elements in `RemoteAccessPoint` and `LocalAccessPoint`. The order of these network addresses dictates the order of preference. The Tuxedo JCA Adapter will attempt using the first network address in `LocalAccessPoint` to establish a listening endpoint. If it fails, it tries the next one until a listening endpoint established, or it exhausted all the network addresses.

The Tuxedo JCA Adapter attempts to use the first network address to establish connection with a remote Oracle Tuxedo `GWTDOMAIN` gateway. If it fails, it tries the next one until either a connection is established, or it exhausted all the network addresses. However, there is no link-level fail back. Listing 40 shows an example using `dmconfig` file configuration that a local access point, `LDOM1`, has two network addresses for link-level failover. It also shows that a remote access point, `RDOM1`, has 2 network addresses for link level fail over.

Listing 40   Link-Level Fail Over Example

```
...
<LocalAccessPoint name="LDOM1">
   <AccessPointId>Godfried</AccessPointId>
   <NetworkAddress>//neocortex:14001</NetworkAddress>
   <NetworkAddress>//cerebrum:14002</NetworkAddress>
</LocalAccessPoint>
<RemoteAccessPoint name="RDOM1">
   <AccessPointId>Geneve</AccessPointId>
   <NetworkAddress>//bluestar:11023</NetworkAddress>
   <NetworkAddress>//orion:11023</NetworkAddress>
</RemoteAccessPoint>
...
```

Besides link-level fail over, Tuxedo JCA Adapter also supports Service-Level Fail Over. The service- level fail over specifies the alternate sessions that the remote Oracle Tuxedo resources can be accessed. This is a comma separated list. It is different from load balancing. Service-Level Fail Over only available when `ConnectionPolicy` equals to `ON_STARTUP`. Other types of `ConnectionPolicy` will only have load balancing.

When Service-level fail over is enabled, it checks the session status of the primary session. If its status is not connected, it checks the first backup session until either a connected status is found for a backup session, or it exhausted all the configured backup session.

Listing 41 shows an example imported resource, `TOUPPER`, load balanced between `session_1` and `session_2`.

Listing 41 Load Balancing Example

```
...
<Import name="TUXTOUPPER">
   <SessionName>session_1</SessionName>
   <SessionName>session_2</sessionName>
   <LoadBalancing>RoundRobin</LoadBalancing>
</Import>
...
```

Listing 42 shows an example that is not only capable of load balancing, but also capable of service-level fail over.

Listing 42 Load Balancing with Service-Level Fail Over Example

```
...
 <Import name="TUXTOUPPER">
   <SessionName>session_1,session_3</SessionName>
   <SessionName>session_2,session_3</sessionName>
```

```
    <LoadBalancing>RoundRobin</LoadBalancing>
</Import>
 ...
```

The above configuration load balances between `session_1` and `session_2`. In the event that `session_1` is not available, the service request is forwarded to `session_3` when the load balancing algorithm decides it is `session_1` turn. The same for `session_2`; in this case session_3 also backs up session_2.

The service-level fail back is automatic when `ConnectionPolicy` is set to `ON_STARTUP` for all the sessions of a particular imported resource. With the previous example, if `session_1` is not available, all service requests destined to `session_1` are routed to `session_3`. When `session_1` becomes available, the service request are routed to the primary route `session_1`.

Oracle Tuxedo JCA Adapter Security

The Tuxedo JCA Adapter supports data security using either Link-Level Encryption or Secured Socket Layer. It also provides outbound identity propagation from application servers to Oracle Tuxedo. This provides finer grain control over Oracle Tuxedo resource access.

The "`DMConfigChecker`" utility is used to ensure security. This utility not only checks the configuration file against the schema, but also converts the password into an encrypted form for better security. If encryption is required, you must run `DMConfigChecker` before starting the Tuxedo JCA Adapter. For more information, see the Oracle Tuxedo JCA Reference Guide.

This section contains the following topics:

- Link-Level Encryption (LLE)
- Secured Socket Layer (SSL) Encryption
- Session Authentication
- Appkey Generator

Link-Level Encryption (LLE)

Link-Level Encryption (LLE), is a fast, proprietary technology that encrypts all user-message flow between the Tuxedo JCA Adapter and the Tuxedo TDomain gateway. It supports 40-bit,

56-bit and 128-bit encryption strength. This feature is enabled by configuring `MaxEncryptBits` and `MinEncryptBits` in the `SessionProfile` of the adapter configuration.

The default value for `MaxEncryptBits` is `128-bit` and the default value for `MinEncryptBits` is 0. The permissible values for both elements are 0 (no encryption), `40-bit`, `56-bit`, `128-bit`, and `256-bit`. `256-bit` encryption is for SSL AES 256-bit encryption (LLE does not support 256-bit encryption). The `MinEncryptBits` value must be smaller than or equal to the `MaxEncryptBits` value.

Notes: If SSL is not configured and `MaxEncryptBits` is set to256-bit, `MaxEncryptBits` is scaled down to maximum 128-bit LLE.

LLE must also be configured in the `GWTDOMAIN` gateway.

Secured Socket Layer (SSL) Encryption

The Tuxedo JCA Adapter also supports Secure Socket Layer (SSL) encryption. To enable SSL encryption, you must do the following:

- Configure Session Profile `MinEncryptBits` and `MaxEncryptBits`.

- Configure Local Access Point `SSLInfo`.

Java Key Store (JKS) is supported. Both `"IdentityKeyStoreFileName"` and `"TrustKeyStoreFileName"` points to the JKS type.

Note: By default, mutual authentication is disabled; however, it can be enabled by setting the `"MutualAuthenticationRequired"` element to `"true"`.

An sample `SSLInfo` configuration example is shown in Listing 43.

Listing 43   SSLInfo Configuration Example

```
<LocalAccessPoint name="jdom">
...
  <SSLInfo>
<IdentityKeyStoreFileName>c:\test\cert\test_users.jks</IdentityKeyStoreFil
eName>
<IdentityKeyStorePassPhraseEncrypted>passphrase</IdentityKeyStorePassPhras
eEncrypted>
    <PrivateKeyAlias>tester</PrivateKeyAlias>
<PrivateKeyPassPhraseEncrypted>passphrase</privatekeypassphraseencrypted>
```

```
    <TrustKeyStoreFileName>c:\test\cert\trusted.jks</TrustKeyStoreFileName
>
<TrustKeyStorePassPhraseEncrypted>passphrase</TrustKeyStorePassPhraseEncry
pted>
  </SSLInfo>
</LocalAccessPoint>
```

In this example, you must run the `DMConfigChecker` utility before it can be used by the Tuxedo JCA Adapter (since there are three elements that require encryption).

Session Authentication

The Tuxedo JCA Adapter supports session authentication when SSL is not configured, but the "Security" of the `SessionProfile` must be configured using "`DM_PW`" or "`APP_PW`".

When the security is configured using "`APP_PW`", it uses the Oracle Tuxedo Application Password as a key to encrypt/decrypt the authenticator. Only one Application Password can be configured for the Tuxedo JCA Adapter. It is configured in the Resources "`ApplicationPasswordEncrypted`" element in the configuration file.

When security is configured with "`DM_PW`", it uses the Domain Password as a key to encrypt/decrypt the authenticator. At most, two passwords pairs can be configured for any given session configured.

Each password pair consists of one local password.one remote password, and their activation/deactivation time. The activation/deactivation time uses the format "`YYYY:MM:DD:hh:mm:ss`" (for example: `2009:01:01:12:00:00`).

The deactivation time must be later than the activation time. If the activation time is not specified, it indicates that the password is already activated when the adapter is booted. If the deactivation time is not specified, then the password never expires.

If a password pair expired while the session is already established, it will not invalidate the session and restarts the session negotiation process. However, if a password pair expired *before* the session negotiation started, then that password pair is not used for authentication. If no valid password pair is found during session authentication, the session cannot be established.

If SSL is required for a Local Access Point, then the session between that Local Access Point and any Remote Access Point uses SSL to underline the data privacy mechanism. In this case, Session

Authentication is not be performed even if "`SessionProfile/Security`" is configured with the correct values.

Appkey Generator

The AppKey Generator is a pluggable class that is used to determine user information to be sent from the application server to Oracle Tuxedo. Oracle Tuxedo uses this information to determine user access rights to an Oracle Tuxedo resource. This is also called ACL.

The Tuxedo JCA Adapter comes preconfigured with a default AppKey Generator class to work with default Oracle Tuxedo Authentication, Authorization, and Auditing plug-in (AAA). However, if Access Control is required using the default Oracle Tuxedo AAA plug-in, then you must also configure the `TpusrFile` element in the Remote Access Point. The `TpusrFile` element in `RemoteAccessPoint` points to an Oracle Tuxedo "`tpusr`" file.

The easiest way to do this is to copy it from Oracle Tuxedo or share it with Oracle Tuxedo (if both are running on the same machine). If the `RemoteAccessPoint TpusrFile` file is not configured, the Oracle JCA Adapter also looks for the `TpusrFile` file in the Resources section.

You must also set "`CredentialPolicy`" to "`Global`" in order to allow the AAA security token to move across the network from the Tuxedo JCA Adapter to an Oracle Tuxedo application domain.

There are other configuration elements in the `RemoteAccessPoint` that further give you the ability to customize the AppKey Generator plug-in. The "`AllowAnonymous`" element tells adapter whether or not anonymous access to Oracle Tuxedo is allowed.

Note: By default, anonymous access is not allowed; the application server performs authentication.

The "`DefaultApplicationKey`" is the key value used by anonymous users to access Oracle Tuxedo (the default value is "`-1`"). The default AppKey Generator assumes the anonymous user name is "`anonymous`".

In order to have successful identity propagation (in addition to all the previously described configuration options), you must also configure the "`Principal Mapping`" information in the host Application Server JCA container.

For more information, see your target application server documentation.

An example Oracle JCA Adapter configuration file identity propagation is shown in Listing 44.

Listing 44   Identity Propagation Example

```
<RemoteAccessPoint name="tdom">
  ...
  <TpusrFile>c:\test\data\tpusr</TpusrFile>
  <AllowAnonymous>true</AllowAnonymous>
  ...
</RemoteAccessPoint>
<SessionProfile name="prof_1">
  ...
  <CredentialPolicy>global</CredentialPolicy>
  ...
</SessionProfile>
```

JATMI Outbound Conversation

- JATMI Conversation Overview
- Using JATMI Outbound Conversation

JATMI Conversation Overview

The Oracle Tuxedo Application-To-Monitor-Interface, also known as ATMI, defines a conversation model that is easy to use. This section describes how to use this Tuxedo JCA Adapter capability. It is intended for use with applications that need to utilize ATMI-style conversation in their application architecture.

ATMI conversation is a Tuxedo JCA Adapter enhancement added to handle a conversation initiated by a Java client (i.e., an outbound conversation between a Java client and an Oracle Tuxedo conversational server).

Note: This product does not support inbound conversations from an Oracle Tuxedo ATMI client.

An Oracle Tuxedo ATMI conversation is a half duplex conversation (which means the sending and receiving is done in a controlled manner). The party that has control of the conversation can send data; when it is time for that party to relinquish the control of the conversation it sends the TPRECVONLY flag along with the data. When the other party receives the latest data, it sees the flag being translated into TPSENDONLY and knows it obtained control the conversation.

A Simple Conversation Description

The Java client initiates the conversation by calling `tpconnect()` to establish a conversation with a remote Oracle Tuxedo conversational server. The Java client continues sending data to a remote conversational server using `tpsend()`. When the Java client finishes sending data, it sends the `TPRECVONLY` flag along with the last `tpsend()` to the server.

When the server receives this data, it turns around to send data back to the Java client until it decides it is time for Java client to send data. These send and receive sequences can be repeated several times until the conversation is completed. When the conversation is completed, the server sends a `tpreturn()`, and the Java client receives an event indicating the conversation is completed.

Conversation Limitation

The upper boundary of control exchanges between a Java client and an Oracle Tuxedo server in any single conversation is `32768`.

Using JATMI Outbound Conversation

The outbound conversation supports the following Oracle Tuxedo ATMI conversational service interfaces for the Java client:

- tpconnect
- tpsend
- tprecv
- tpdiscon

The change of conversation control is done using a pair of flags for `tpconnect`, `tpsend`, and the `tprecv` method. These flags are defined in the `weblogic.wtc.jatmi.TPException` class. These flags are:

- TPSENDONLY
- TPRECVONLY

The Java client initiates conversation using the `tpconnect()` extension from the Tuxedo JCA Adapter implementation of the Interaction class, `TuxedoInteraction`. The `TuxedoInteraction` class `tpconnect()` method returns a conversational object if the conversation is started successfully. The Java client can then use the returned conversational

object to continue communicating with an Oracle Tuxedo conversational server using `tpsend()` and `tprecv()` object interface.

For a normal conversation to complete, the Oracle Tuxedo conversational server must have control over the conversation. It ends the conversation normally through `tpreturn()` in the service routine. In this case, the Java client should receive a `TPReplyException` with conversational event "`TPEV_SVCSUCC`" (or "`TPEV_SVCFAIL`").

The JATMI Conversation model uses `TPReplyExcepton` to carry the conversational event. When the conversational event needs to be conveyed to a Java client, the Tuxedo JCA Adapter throws `TPReplyException`.

`TPReplyException` can also carry data returned from an Oracle Tuxedo conversational server. When a Java client captures a `TPReplyException`, it should also check both event and data. There are two types of conversational events:

- conversation completion status

- conversation change of control.

`TPEV_DISCONIMM`, `TPEV_SVCSUCC`, `TPEV_SVCERR`, `TPEV_SVCFAIL` are conversation completion status types. When any one of these events occur, the conversation has already become stale and the conversation object should not be used. `TPEV_SENDONLY` is a conversation change of control event type; When this event occurs the Java client regains control of the conversation.

If there is no event (whether there is accompanied data or not), the `tprecv()` performs a normal return without any exception. If there is an error encountered related to connection, protocol, and configuration, the Tuxedo JCA Adapter throws a `TPException`. In the above illustration, the Oracle Tuxedo conversation server initiates a `tpreturn` (`TPSUCCESS`) and the client initiates a `tprecv()` and retrieves the `TPEV_SVCSUCC` event from `TPReplyException`.

JATMI Conversation Extension

A new method, (`tpconnect()`used to initiate a conversation with an Oracle Tuxedo Conversational serve), r is added to the `com.oracle.tuxedo.adapter.cci.TuxedoInteraction` class. To access this new ATMI conversation, you must typecast the Interaction object returned from `Connection.createInteraction()` to `TuxedoInteraction` class to gain access to the JATMI conversation extension.

shows a `TuxedoInteraction` class `tpconnect` interface definition example.

Listing 45   TuxedoInteraction Class tpconnect Interface Definition

```
…

import weblogic.wtc.jatmi.Conversation;

…

public Conversation tpconnect(String svcname, TypedBuffer data, int flags)
throws TPException;

…
```

Either `TPSENDONLY` or `TPRECVONLY` must be specified for the flags field in `tpconnect()`, these
two flags are defined in the `weblogic.wtc.jatmi.ApplicationToMonitorInterface` class
as shown in Listing 46.

- `TPSENDONLY`

  Caller wants the conversation to be set up initially so that it only sends data The called
  Oracle Tuxedo service only receives data.

- `TPRECVONLY`

  Caller wants the conversation to be set up initially so that it only receives data and the
  called Oracle Tuxedo service only sends data.

Listing 46   weblogic.wtc.jatmi.ApplicationToMonitorInterface Class

```
import weblogic.wtc.jatmi.TPException;

import weblogic.wtc.jatmi.TPReplyException;

import weblogic.wtc.jatmi.ApplicationToMonitorInterface;

import weblogic.wtc.jatmi.Conversation;

import weblogic.wtc.jatmi.Reply;

import com.oracle.tuxedo.adapter.cci.TuxedoInteraction;


try {

    myTux = tcf.getConnection();
```

```
    ix        = (TuxedoInteraction)myTux.createInteraction();

    myConversation = ix.tpconnect("CONVSVC", myData,
ApplicationToMonitorInterface.TPSENDONLY);

}

Catch (TPException tpe) {

  …

}
…
```

JATMI Conversation Object

The `tpconnect()` interface in the `com.oracle.tuxedo.adapter.cci.TuxedoInteraction` class returns a `weblogic.wtc.jatmi.Conversation` conversational object. You can use the returned conversational object to have a conversation with an Oracle Tuxedo conversational server.

Listing 47 is an interface declaration example.

Listing 47   Interface Declaration Example

```
public void tpsend(TypedBuffer data, int flags) throws TPException;

public Reply tprecv(int flags) throws TPException, TPReplyException;

public void tpdiscon() throws TPException;
```

Interface Method tpsend

Public void `tpsend(TypedBuffer data, int flags)` throws `TPException`.This method sends data across an open conversation from a Java client to an Oracle Tuxedo conversational server. The Java client must have control of the conversation. If the Java client initiates `tpsend()` in an improper context (i.e., without conversation control), it receives a `TPException.TPEPROTO` error.

The flags field are as follows:

- TPRECVONLY

  This flag signifies that after caller data is sent, the caller gives up control of the conversation; this means the caller can no long issue any more `tpsend` calls. When the receiver on the other end of conversation receives the data, it l also receives an event, `TPEV_SENDONLY`, indicating that it has conversation control.

  If the java client wants to send data several times before the last data sent with the `TPRECVONLY` flag, the client can set the flags value to 0.

When an error occurs, a `TPException` is thrown and the error code can be retrieved by calling `TPExcepiton.gettperrno()`. The following is the list of possible `TPException` errors:

- TPEINVAL

  Invalid arguments were given.

- TPETIME

  A timeout occurred.

- TPEEVENT

  An event occurred. User data is not sent when this error occurs. The event type is returned in event, and it can be retrieved by calling `TPReplyException.getrevent()`.

- TPEPROTO

  `tpsend()` was called improperly (e.g., without conversation control).

- TPESYSTEM

  Either the Tuxedo JCA Adapter or the remote access point encountered a problem.

Interface Method tprecv

Public Reply `tprecv(int flags)` throws `TPException`, `TPReplyException`. This method is used by a Java client to receive data sent by an Oracle Tuxedo conversational server across an open conversation. The method call can only be issued by a Java client when it does not have conversation control. If the Java client calls this method improperly (i.e., still has conversation control), a `TPException.TPEPROTO` exception is thrown. This method can throw either `TPException` or `TPReplyException`, depending on the nature of the situation.

Normally, it uses `TPReplyException` to convey the event coming from the Oracle Tuxedo conversation server (for example, conversation completion, or change of conversation control). However, sometimes it can also throw `TPException` because an unexpected error is encountered while receiving data (for example, calling this method improperly or encountering a network error).

If an event exists for the conversation, `tprecv()` returns `TPReplyException` with `TPERRNO` set to `TPException.TPEEVENT`. The event type is returned as well, and the Java client can retrieve it through invoking `TPReplyException.getrevent()`. The following is a list of valid events.

- `TPEV_SENDONLY`

  The Oracle Tuxedo conversational server relinquished control of the conversation. The Java client is allowed to send data, but cannot receive any data until it relinquishes control.

- `TPEV_SVCERR`

  This event indicates that the Oracle Tuxedo conversational server has initiated `tpreturn()` and during `tpreturn()` it encountered an error that precluded the service from returning successfully. Because of the nature of this event, any application defined data or return code are not available. The conversation also becomes invalid.

- `TPEV_SVCFAIL`

  This event indicates that the Oracle Tuxedo conversation server has finished unsuccessfully as defined by the application. It called `tpreturn()` with either `TPFAIL` or `TPEXIT`. If the Oracle Tuxedo conversation server has conversation control at the time `tpreturn()` was called, then it can pass an application defined return code and a typed buffer. When the Java client sees this event, the conversation becomes invalid.

- `TPEV_SVCSUCC`

  This event indicates that the Oracle Tuxedo conversational server has finished the conversation successfully as defined by the application and calls `tpreturn()` with `TPSUCCESS`. When the Java client receives this event, the conversation is invalid.

- `TPEV_DISCONIMM`

  This event indicates that originator of the conversation has issued an immediate connection disconnect via `tpdiscon()`. This event is also returned to the originator or subordinate when a connection is broken due to a communication error. In regards to the Tuxedo JCA Adapter, this could also be due to the server initiating `tpreturn()` without conversation control; the TDOMAIN protocol translates it into a `TPEV_DISCONIMM` event. Because this is an immediate disconnection notification abortive (rather than orderly), data in transit may be lost. The descriptor used for the connection is no longer valid.

When an error occurs, a `TPException` is thrown and the error code can be retrieved by calling `TPExcepiton.gettperrno()`. The following is a list of possible `TPException` errors:

- `TPEINVAL`

  Invalid arguments were given.

- `TPETIME`

  A timeout occurred.

- `TPEEVENT`

  An event occurred. Its event type is available in event. It can be retrieved by calling `TPReplyException.getrevent()`.

- `TPEPROTO`

  `tprecv()` was called improperly.   For example, the Java client still has conversation control.

- `TPESYSTEM`

Either the Tuxedo JCA Adapter or a remote access point encountered a problem.Java client code should capture both exception types and handle them accordingly.For a Java client conversation always ending with `tprecv()`, the last `tprecv()` corresponds to the Oracle Tuxedo conversation server `tpreturn()`. When a Java client is done using the Connection that obtained through `ConnectionFactory.getConnection()`, it must return the Connection to the pool by calling `Connection.close()`; failing to do this causes a Connection leak.

Interface Method tpdiscon

public void `tpdiscon()` throws `TPException`;

The `tpdiscon()` method immediately tears down the conversation represented by the conversational object. This method can only be called by the initiator of the conversation (which is the Java client). Although a conversational Java client can tear down a conversation using `tpdiscon()`, it is preferred to let the Oracle Tuxedo conversational server tear down the connection using `tpreturn()`; doing so ensures correct results.

Conversation Control

Relinquish Conversation Control

When a conversation event occurs, Java clients receive a `TPReplyException`. Java clients must call `TPRecplyException.gettperrno()` to decide whether an event occurred or not ,and then uses `TPReplyException.getrevent()` to retrieve the conversational event.

shows a relinquish conversation control example.

Listing 48   Relinquish Conversation Control

```
…
try {
   myConversation.tpsend(myData, ApplicationToMonitorInterface.TPRECVONLY);
}
Catch (TPException tpe) {
   …
}
…
…
try {
   myConversation.tpsend(myData, ApplicationToMonitorInterface.TPRECVONLY);
}
Catch (TPException tpe) {
   …
}
…
```

Regain Conversation Control

The Tuxedo JCA Adapter conversational Java client can regain conversation control when it receives a TPEV_SENDONLY event inside a TPReplyException while calling tprecv(). The Java client must check the returned TPERRNO to see if the conversational event exists or not, and then retrieves the event as shown in Listing 49.

Listing 49   Regain Conversation Control

```
while (…) {
   Try {
```

```
  rply = myConversation.tprecv(0);
}
catch (TPReplyException tre) {
  switch (tre.gettperrno()) {
  case TPException.TPEEVENT:
    switch (tre.getrevent()) {
    case TPException.TPEV_SENDONLY:
```
…

Retrieving Conversational Event

The Oracle Tuxedo conversational server also could regain conversation control when it receives a TPEV_SENDONLY event while calling tprecv(). It also could relinquish the conversational control by calling tpsend() with TPRECVONLY.

Listing 50   Retrieving Conversational Event

```
try {
  myRtn  = myConversation.tprecv();
  }
Catch (TPReplyException tre) {
  Switch (tre.gettperrno()) {
  Case TPException.TPEEVENT:
    Switch (tre.getrevent() {
    Case TPException.TPEV_SENDONLY:
      /* got the control of the conversation */
    …
    }
  …
```

```
    }
}
```

Configuration

There is no special configuration required to make a conversation; however, a correctly configured Oracle Tuxedo conversation server must be presented in TUXCONFIG and exports its service through BDMCONFIG. For the Tuxedo JCA Adapter, you can elect not to configure Import and allow the default import to work, or you can configure an Import for every imported Oracle Tuxedo service (including conversational service).

Oracle Tuxedo Configuration

An Oracle Tuxedo server must be configured as a Conversational Server as shown in Listing 51.

Listing 51   Conversational Server

```
*SERVERS
DEFAULT:
        CLOPT="-A" RESTART=Y MAXGEN=5
Convserv     SRVGRP=GROUP4 SRVID=41 CONV=Y
```

In the above example, the Oracle Tuxedo server "Convserv" is a conversational server set by "CONV=Y".

A complete Oracle Tuxedo configuration file will be given in "Sample Conversation Application"

Oracle Tuxedo Domain Configuration

The conversation service must be exported by Tuxedo GWTDOMAIN using the configuration in the BDMCONFIG file. For example, if "Convserv" provides the service "CTOUPPER", then "CTOUPPER" must be exported through the BDMCONFIG file.

```
*DM_LOCAL_SERVICES
```

```
CTOUPPER
```

Oracle Tuxedo JCA Adapter Configuration

The same service exported by the Oracle Tuxedo Domain must be imported by the Tuxedo JCA Adapter. There are two choices:

- You can either explicitly configure the Oracle Tuxedo service (i.e., CTOUPPER), as an imported resource, or

- Do not configure any imported resource.

If you choose not to configure any imported resource, then the Tuxedo JCA Adapter default Import feature is initiated. In this case the Tuxedo JCA Adapter forwards the service request to the partner Oracle Tuxedo Domain through the GWTDOMAIN gateway regardless of whether the external resource requested by the Java client actually exists or not.

If you chooses to configure imported resource, then the Tuxedo JCA Adapter only allows service requests that call those explicitly configure external resources. Any service (usually an Oracle Tuxedo service), that requests a non-configured external resource is rejected and throws a TPException.TPENOENT error.

Listing 52 shows a Tuxedo JCA Adapter `dmconfig` file imported resource example.

Listing 52   Tuxedo JCA Adapter dmconfig File Imported Resource

```
…
<Import name="CTOUPPER">

  <RemoteName>CTOUPPER</RemoteName>

  <SessionName>session_1</SessionName>

  <LoadBalancing>RoundRobin</LoadBalancing>

</Import>
…
```

Example Conversation Application

Listing 53 shows a very simple complete example application.

1. The client initiates `tpconnect()` with data to start conversation.

2. The client calls `tpsend()` to send a second piece of data and gives control to the Oracle Tuxedo Server.

3. The Oracle Tuxedo server initiates `tpsend()` and then calls `tpreturn()` to end conversation.

Oracle Tuxedo Server Application

This is the Oracle Tuxedo Conversation server part of the application.

The name of the source file is called "`convsimp.c`".

Listing 53   Oracle Tuxedo Conversation Server

```
Oracle Tuxedo Server Applicatiov/*
 * Copyright (c) 2012 by Oracle. All Rights Reserved.
 */
#include <tmenv.h>
#include <stdio.h>
#include <ctype.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <atmi.h>/* TUXEDO Header File */
#include <userlog.h>/* TUXEDO Header File */
#include <Uunix.h>/* TUXEDO Header File */


#if defined(__STDC__) || defined(__cplusplus)
tpsvrinit(int argc, char *argv[])
#else
tpsvrinit(argc, argv)
```

```
int argc;

char **argv;

#endif

{

tpopen();

        userlog("tpsvrinit(10) Welcome to the conversational simple
server");

        return(0);

}


#ifdef __cplusplus

extern "C"

#endif

void

#if defined(__STDC__) || defined(__cplusplus)

MYCONV(TPSVCINFO *rqst)

#else

MYCONV(rqst)

TPSVCINFO *rqst;

#endif

{

        long    revent;

        long    the_len;

        long    ret;

        long    flags;

        char*   line = NULL;


        userlog("> MYCONV()");
```

```
        if ((line = tpalloc("STRING", "", 1024)) == (char *)NULL) {

                userlog("< MYCONV(10) failed to allocate STRING buffer");

                tpreturn(TPFAIL, 0, NULL, 0, 0);

        }


         the_len = rqst->len;
         userlog("receive rqst->data %p, the len = %d", rqst->data, the_len);
              if (rqst->data != NULL && the_len > 0) {
              userlog("recv data #1: %s", rqst->data);
          }


        if (rqst->flags & TPSENDONLY) {

                userlog("rqst->flags & TPSENDONLY");

                goto send_data;

        }


        userlog("call tprecv()");
    if (tprecv(rqst->cd, &rqst->data, &the_len, TPNOCHANGE, &revent) != -1) {
                              userlog("receive rqst->data %p, the len = %d",
rqst->data, the_len);
                if (rqst->data != NULL) {
                        userlog("recv data #2: %s", rqst->data);
              }
        }
        /* check for event */
        if ((tperrno == TPEVENT) && (revent == TPEV_SENDONLY)) {

                userlog("recv revent = TPEV_SENDONLY");
```

```
        }

        else {

                /* this is not what I expected */

                userlog("< MYCONV(20) invalid errno %d(%s) or event 0x%lx",

                        tperrno, tpstrerror(tperrno), revent);

                tpreturn(TPFAIL, 1, rqst->data, rqst->len, 0);

        }

        userlog("1st tprecv() successful");


send_data:

        userlog("start 1st tpsend()");

        (void)strcpy(line, "return data #1");

        revent = 0;

        if (tpsend(rqst->cd, line, strlen(line)+1, 0, &revent) == -1) {

                (void) userlog("MYCONV(20) tpsend error");

                if(tperrno == TPEEVENT) {

                        userlog("< MYCONV(30) got event: 0x%lx", revent);

                }

                else {

                        userlog("< MYCONV(40) tperrno = %d (%s)",

                                tperrno, tpstrerror(tperrno));

                }

                /* this is not what I expected */

                tpreturn(TPFAIL, 2, rqst->data, rqst->len, 0);

        }

        userlog("1st tpsend() successful");
```

```
        userlog("< MYCONV() return(50) TPSUCCESS");

        (void)strcpy(line, "my final data");

        tpreturn(TPSUCCESS, 0, line, strlen(line) + 1, 0);

}
```

Build Tuxedo Conversation Server

There is no difference between building an Oracle Tuxedo conversational server and a regular
server. To build the above Oracle Tuxedo conversational server, issue the following command:

```
buildserver -f convsimp.c -o convsimp  -s MYCONV
```

Oracle Tuxedo Configuration File

Listing 54 is an example Oracle Tuxedo configuration file.

Listing 54   Oracle Tuxedo Configuration File

```
#Ubbconfig of Tuxedo Domain
#


*RESOURCES
IPCKEY              51301
MASTER              site1
MAXACCESSERS100
MAXSERVERS    25
MAXSERVICES    50
MODEL                      SHM
LDBAL                      N
BLOCKTIME 10
SCANUNIT        5
```

```
SECURITY        NONE


*MACHINES

DEFAULT:

            APPDIR="C:\test\JCA\tdom"

            TUXCONFIG="C:\test\JCA\tdom/TUXCONFIG"

            TUXDIR="c:\tuxedo\Tuxedo11gR1PS1"


"NEOCORTEX"LMID=site1

            MAXWSCLIENTS=2


*GROUPS

GROUP1 LMID=site1 GRPNO=1OPENINFO=NONE

GROUP2 LMID=site1 GRPNO=2OPENINFO=NONE

GROUP3 LMID=site1 GRPNO=3OPENINFO=NONE

*SERVERS

DEFAULT:

            CLOPT="-A" RESTART=Y MAXGEN=5


DMADM   SRVGRP=GROUP1SRVID=21

GWADM   SRVGRP=GROUP2SRVID=31

GWTDOMAINSRVGRP=GROUP2SRVID=32

convsimpSRVGRP=GROUP3SRVID=41 CONV=Y


*SERVICES

MYCONV
```

Oracle Tuxedo Domain Configuration

Listing 55 shows an Oracle Tuxedo Domain configuration example.

Listing 55   Oracle Tuxedo Domain Configuration

```
#
*DM_RESOURCES
#
VERSION=U22
#


#
#
*DM_LOCAL_DOMAINS
#
#
"TDOM"    GWGRP=GROUP2
        TYPE=TDOMAIN
        DOMAINID="TDOM_ID"
        BLOCKTIME=60
        SECURITY=NONE


#
*DM_REMOTE_DOMAINS
#
#
JDOM    TYPE=TDOMAIN
        DOMAINID="JDOM_ID"
```

```
#
#
*DM_TDOMAIN
#
TDOM    NWADDR="//localhost:12478"
JDOM    NWADDR="//localhost:10801"
#
#
*DM_LOCAL_SERVICES
#
#Exported
#
MYCONV
#
*DM_REMOTE_SERVICES
#
#Imported
#
```

Java Client Application

The Java client is implemented here as a Java Session Bean utilizing the Tuxedo JCA Adapter JATMI extension to have a conversation with the Oracle Tuxedo Conversation server.   The Connection and Interaction must be closed by the Java client when processing is done, no matter whether the conversation is successful or not; failing to do so will leak the Connection.

Listing 56 shows a JATMI client code example for WebLogic server.

Listing 56   Java Client Application

```
package test.conv.convsimp;


import javax.naming.Context;

import javax.naming.InitialContext;

import javax.naming.NamingException;

import javax.ejb.CreateException;


import javax.resource.cci.ConnectionFactory;

import javax.resource.cci.Connection;

import javax.resource.cci.Interaction;

import javax.resource.cci.InteractionSpec;

import javax.resource.ResourceException;


import weblogic.ejb.GenericSessionBean;

import weblogic.wtc.jatmi.Reply;

import weblogic.wtc.jatmi.TPException;

import weblogic.wtc.jatmi.TPReplyException;

import weblogic.wtc.jatmi.TypedString;

import weblogic.wtc.jatmi.TypedCArray;

import weblogic.wtc.jatmi.CallDescriptor;

import weblogic.wtc.jatmi.ApplicationToMonitorInterface;

import weblogic.wtc.jatmi.Conversation;

import weblogic.ejbgen.*;


/* the part with Tuxedo JCA Adapter */

import com.oracle.tuxedo.utils.TuxedoLogger;
```

```
import com.oracle.tuxedo.adapter.cci.TuxedoStringRecord;

import com.oracle.tuxedo.adapter.cci.TuxedoInteractionSpec;

import com.oracle.tuxedo.adapter.cci.TuxedoConnectionFactory;

import com.oracle.tuxedo.adapter.cci.TuxedoJCAConnection;

import com.oracle.tuxedo.adapter.cci.TuxedoInteraction;


/**

 * TuxConversationBean is a stateful SessionBean. This EJBean illustrates:

 * <ul>

 * <li> The use of the Tuxedo JCA connector.

 * <li> The use of Application-defined exceptions.

 * <li> The use of JATMI conversational extension with Tuxedo Server.

 * </ul>

 *

 * Copyright (c) 2012 by Oracle. All Rights Reserved.

 */

@FileGeneration(remoteClass = Constants.Bool.TRUE,

                localHome = Constants.Bool.FALSE,

                remoteHome = Constants.Bool.TRUE,

                remoteClassName = "TuxConversation",

                remoteHomeName = "TuxConversationHome",

                localClass = Constants.Bool.FALSE)

@JarSettings(ejbClientJar = "convsimp_client.jar")

@JndiName(remote = "tuxedo.services.TuxConversationHome")

@Session(idleTimeoutSeconds = "600",

        maxBeansInCache = "100",

        transTimeoutSeconds = "0",
```

```
         type = Session.SessionType.STATEFUL,

         defaultTransaction = Constants.TransactionAttribute.SUPPORTS,

         ejbName = "TuxConversation")
public class TuxConversationBean extends GenericSessionBean {

  static final boolean VERBOSE = true;


  /**
   * This implementation

   * will get the TuxedoConnectionFactory from JNDI, and use it to get

   * an Oracle Tuxedo object, which can then be used to do the actual
tpconnect.

   * TypedString object.

   * @param cmd A string to be used as command or first data item (not null)

   * @param target A string to be used for that target name or send data
itme. (not null)

   * @return a string from server

   */
  @RemoteMethod()
  public String Conversation(String cmd, String target) throws TPException,
TPReplyException
  {
    String ret = null;
    try {
      ret = doConversation(cmd, target);
    }
    catch (TPReplyException tre) {
      log("TPReplyException: " + tre);
```

```
      throw tre;

    }

    catch (TPException te) {

      log("TPException: " + te);

      throw te;

    }

    catch (Exception e) {

      log("Exception: " + e);

      throw new TPException(TPException.TPESVCFAIL, e.getMessage());

    }

    return ret;

  }


  private String doConversation(String cmd, String target) throws
TPException, TPReplyException, Exception

  {

    Context                 ctx;

    TuxedoConnectionFactory tcf;

    TuxedoJCAConnection     myTux;

    TypedString             myData     = null;

    TypedString             returnData1 = null;

    TypedString             returnData2 = null;

    Reply                   myRtn      = null;

    int                     flags;

    Conversation            myConversation;

    TuxedoInteraction       ix;


    log("cmd: " + cmd +  ", target: " + target);
```

```
    try {

      ctx = new InitialContext();

      tcf =
  (TuxedoConnectionFactory)ctx.lookup("eis/TuxedoConnectionFactory");

    }

    catch (Exception ne) {

      ne.printStackTrace();

      log("eis/TuxedoConnectionFactory lookup failed");

      throw new TPException(TPException.TPENOENT, "Could not get
  eis/TuxedoConnectionFactory:" +

                            ne.getMessage());

    }


    /*
     * Get connection and interaction
     */

    myTux = null;

    ix    = null;

    try {

      myTux = (TuxedoJCAConnection)tcf.getConnection();

      ix    = (TuxedoInteraction)myTux.createInteraction();

    }

    catch (ResourceException re) {

      re.printStackTrace();

      doClose(myTux, ix);

      log("failed to create interaction");
```

```
    throw new TPException(TPException.TPENOENT, "Could not create
interaction:" +
                          re.getMessage());
    }


    myData = new TypedString(cmd);


    if (cmd.equalsIgnoreCase("DoReceiveOnly")) {
      flags = ApplicationToMonitorInterface.TPRECVONLY;
      log( "set flags = TPRECVONLY");
    }
    else {
      flags = ApplicationToMonitorInterface.TPSENDONLY;
      log("set flags = TPSENDONLY");
    }
    /*
     * Get Conversation object.
     */
    try {
      myConversation = ix.tpconnect("MYCONV", myData, flags);
    }
    catch (TPException te) {
      doClose(myTux, ix);
      te.printStackTrace();
      log("get conversation object failure");
      throw te;
    }
    catch (Exception ee) {
```

```
        doClose(myTux, ix);

        ee.printStackTrace();

        log(" get conversation object failure");

        throw new TPException(TPException.TPESYSTEM,

                             "failed to get conversation object:  " +
   ee.getMessage());

     }

     log("tpconnect successful!");


     /*
      * first tpsend(), but 2nd data send.
      */

     if (flags == ApplicationToMonitorInterface.TPSENDONLY) {

        log("first tpsend()");

        myData = new TypedString(target);

       try {

          myConversation.tpsend(myData,
   ApplicationToMonitorInterface.TPRECVONLY);

       }

       catch (TPException te) {

          te.printStackTrace();

          doClose(myTux, ix);

          log("TPException: " + te);

          throw te;

       }

       log("tpsend successful!");

     }
```

```
/*
 * tprecv() first part of data.
 */
try {
  log("1 call tprecv()");
  myRtn = myConversation.tprecv(0);
}
catch (TPReplyException tre) {
  switch (tre.gettperrno()) {
  case TPException.TPEEVENT:  /* check for conversation event */
     switch (tre.getrevent()) { /* handle according to type of event */
     case TPException.TPEV_SENDONLY:
       log("receive TPEV_SENDONLY");
       /* not expecting change of direction */
       doClose(myTux, ix);
       log(" unexpected SENDONLY event: " + tre);
       throw new TPException(TPException.TPESYSTEM,
                            "Unexpected SENDONLY event:" + tre);
     default:
         // anything else in a RECEIVE state is an error
         doClose(myTux, ix); /* needs to close connection */
         log("unexpected event from server: " + tre);
         throw new TPException(TPException.TPESYSTEM,
                              "Unexpected tprecv event:" + tre);
     }  /* end of switch event type */
  default:
    doClose(myTux, ix); /* needs to close connection */
```

```
      log("got unexpected error from server: " + tre);

      throw new TPException(TPException.TPESYSTEM,

                            "Unexpected tprecv exception:" + tre);

   } /* end of switch tperrno type */

}

catch (TPException te) {

  te.printStackTrace();

  doClose(myTux, ix); /* needs to close connection */

  log("tprecv threw TPException: " + te);

  throw te;

}

catch (Exception ee) {

  ee.printStackTrace();

  doClose(myTux, ix);

  log("tprecv threw Exception: " + ee);

  throw new TPException(TPException.TPESYSTEM, "Exception: " + ee);

}

/*

 * handles returned data

 */

returnData1 = (TypedString)myRtn.getReplyBuffer();

log("first receive successful!");


/*

 *2nd tprecv(), server does a tpreturn()

 */

try {
```

```
    log("2 call tprecv()");

    myRtn = myConversation.tprecv(0);

}

catch (TPReplyException tre) {

    switch (tre.gettperrno()) {

    case TPException.TPEEVENT:  /* check for conversation event */

        switch (tre.getrevent()) { /* handle according to type of event */

        case TPException.TPEV_SVCSUCC:

            log("receive TPEV_SVCSUCC");

            myRtn = tre.getExceptionReply();

            break;

        case TPException.TPEV_SVCFAIL:

            doClose(myTux, ix); /* needs to close connection */

            log("unexpected event from server: " + tre);

            throw new TPException(TPException.TPESYSTEM,

                            "Unexpected tprecv event TPEV_SVCFAIL:" + tre);

        case TPException.TPEV_DISCONIMM:

            doClose(myTux, ix); /* needs to close connection */

            log("unexpected event from server: " + tre);

            throw new TPException(TPException.TPESYSTEM,

                        "Unexpected tprecv event TPEV_DISCONIMM:" + tre);

        default:

            // anything else in a RECEIVE state is an error

            log("event = " + tre.getrevent());

            doClose(myTux, ix);

            log("unexpected event from server: " + tre);

            throw new TPException(TPException.TPESYSTEM,
```

```
                                  "Unexpected tprecv event:" + tre);
        }  /* end of switch event type */
        break;
    default:
      doClose(myTux, ix);  /* needs to close connection */
      log("got unexpected error from server: " + tre);
      throw new TPException(TPException.TPESYSTEM,
                            "Unexpected tprecv exception:" + tre);
    } /* end of switch tperrno type */
  }
  catch (TPException te) {
      doClose(myTux, ix);  /* needs to close connection */
      te.printStackTrace();
      log("tprecv threw TPException: " + te);
      throw te;
  }
  catch (Exception ee) {
      doClose(myTux, ix); /* needs to close connection */
      ee.printStackTrace();
      log("tprecv threw Exception: " + ee);
      throw new TPException(TPException.TPESYSTEM, "Exception: " + ee);
  }
  returnData2 = (TypedString)myRtn.getReplyBuffer();
  log("second receive successful!");


  doClose(myTux, ix);  /* needs to close connection even when everything
is good */
  log("conversation successful");
```

```
    return returnData1.toString() + returnData2.toString();

}


private void doClose(Connection conn, Interaction ix)

{

  try {

    if (ix != null) {

      ix.close();

      ix = null;

    }

    if (conn != null) {

      conn.close();

      conn = null;

    }

  }

  catch (Exception e) {

   log("Exception occurred while closing connection! " + e.getMessage());

  }

}


/**

 * This method is required by the EJB Specification,

 * but is not used by this example.

 */

public void ejbCreate() throws CreateException { }

public void ejbPostCreate() throws CreateException { }
```

```
  private void log(String s) {

    if (VERBOSE) {

      System.out.println(s);

    }

  }

}
```

Oracle Tuxedo JCA Adapter Configuration

The dmconfig File

The `dmconfig` file is a property in the Resource Adapter Deployment Descriptor that tells the Tuxedo JCA Adapter where to look for its configuration file as shown in Listing 57.

Listing 57   dmconfig File

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

<TuxedoConnector>

  <LocalAccessPoint name="JDOM">

    <AccessPointId>JDOM_ID</AccessPointId>

    <NetworkAddress>//localhost:10801</NetworkAddress>

  </LocalAccessPoint>

  <RemoteAccessPoint name="TDOM">

    <AccessPointId>TDOM_ID</AccessPointId>

    <NetworkAddress>//localhost:12478</NetworkAddress>

  </RemoteAccessPoint>

  <SessionProfile name="profile_1">

    <BlockTime>60000</BlockTime>

    <ConnectionPolicy>ON_STARTUP</ConnectionPolicy>
```

```
  </SessionProfile>

  <Session name="session_1">

    <LocalAccessPointName>JDOM</LocalAccessPointName>

    <RemoteAccessPointName>TDOM</RemoteAccessPointName>

    <ProfileName>profile_1</ProfileName>

  </Session>

  <Import name="MYCONV">

    <RemoteName>MYCONV</RemoteName>

    <SessionName>session_1</SessionName>

    <LoadBalancing>RoundRobin</LoadBalancing>

  </Import>

</TuxedoConnector>
```

The above configuration example configures one local access point "JDOM", one remote access point "TDOM". It also configures one Oracle Tuxedo TDomain session between JDOM and TDOM named "session_1", and also configures session_1 profile with the name "profile_1".

There is one imported resource configured with the name "MYCONV" corresponding with the Oracle Tuxedo exported service name "MYCONV".

The Resource Adapter Deployment Descriptor

Listing 58 shows the corresponding Resource Adapter Deployment Descriptor. This file is named "ra.xml", and is located in the META-INF of the Resource Archive file.

Listing 58   The Resource Adapter Deployment Descriptor

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

<connector xmlns="http://java.sun.com/xml/ns/j2ee"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd"
```

```
     version="1.5">

   <display-name>Tuxedo JCA Adapter</display-name>

   <vendor-name>Oracle</vendor-name>

   <eis-type>Tuxedo</eis-type>

   <resourceadapter-version>12gR1(12.1.1.0.0)</resourceadapter-version>

   <license>

     <description>Tuxedo SALT license</description>

     <license-required>false</license-required>

   </license>

   <resourceadapter>


<resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoResourceAdapter</re
sourceadapter-class>

     <!--


<resourceadapter-class>com.oracle.tuxedo.adapter.TuxedoFBCResourceAdapter<
/resourceadapter-class>
       The following is the list of properties name can be configured as
adapter-wise configuration.

       traceLevel  - java.lang.String  - a numerical value

       xaAffinity  - java.lang.String  - transaction affinity to a remote
domain, "true" or "false", default to true

       keyFileName - java.lang.String  - encryption key file name

       throwFailureReplyException - java.lang.Boolean - default to ture

       appManagedLocalTxTimeout   - java.lang.Integer - Application managed
transaction or AUTOTRAN timeout

                                                       defaults to 300 seconds

     fieldTable16Class - java.lang.String - a comma-separated list of fully
qualified FML classes
```

```
    fieldTable32class - java.lang.String - a comma-separated list of fully
qualified FML32 classes

    viewFile16Class   - java.lang.String - a comma-separated list of fully
qualified VIEW classes

    viewFile32Class   - java.lang.String - a comma-separated list of fully
qualified VIEW32 classes

     tpusrFile          - java.lang.String - path name to the TPUSR file

    remoteMBEncoding  - java.lang.String - remote Tuxedo encoding name for
multi-byte language

    mBEncodingMapFile - java.lang.String - path name to Multi-byte encoding
name mapping

     autoTran          - java.lang.Boolean- enable adapter-wise AUTOTRAN,
default to false

    -->

    <config-property>

      <config-property-name>dmconfig</config-property-name>

      <config-property-type>java.lang.String</config-property-type>


<config-property-value>C:\test\JCA\adapter\dmconfig.xml</config-property-v
alue>

    </config-property>

    <config-property>

      <config-property-name>debugConfig</config-property-name>

      <config-property-type>java.lang.Boolean</config-property-type>

      <config-property-value>true</config-property-value>

    </config-property>

    <outbound-resourceadapter>

      <connection-definition>
```

```
<managedconnectionfactory-class>com.oracle.tuxedo.adapter.spi.TuxedoManage
dConnectionFactory</managedconnectionfactory-class>

        <!--

          The following is the list of properties that user can use them

          to configure connection pool or connection factory.

          User must either configure localAccessPointSpec or

          connectionFactoryName if transaction is used.

        These property described here is serving as template, user should not

          configure them here, instead user should configure them either
through WebSphere console

          or weblogic-ra.xml side file.

        -->

        <config-property>

          <description>factory-wise AUTOTRAN setting, default to false,
overrides adapter-wise setting</description>

          <config-property-name>autoTran</config-property-name>

          <config-property-type>java.lang.Boolean</config-property-type>

        </config-property>

        <config-property>

          <description>factory-wise Failure Reply Exception setting, default
to true, overrides adapter-wise setting</description>

<config-property-name>throwFailureReplyException</config-property-name>

          <config-property-type>java.lang.Boolean</config-property-type>

        </config-property>

        <config-property>

          <description>factory-wise application managed transaction or
AUTOTRAN time out, overrides adapter-wise setting</description>
```

```
<config-property-name>appManagedLocalTxTimeout</config-property-name>

        <config-property-type>java.lang.Integer</config-property-type>

      </config-property>

      <config-property>

       <description>connection factory or pool name, this is required if
XA or local application managed

                     transaction is required</description>

      <config-property-name>connectionFactoryName</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

      </config-property>

      <config-property>

        <description>application password in either clear text or cipher
text using com.oracle.tuxedo.tools.EncryptPassword tool</description>

        <config-property-name>applicationPassword</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

      </config-property>

      <config-property>

        <description>local access point specification of the format
//hostname:port/domainId=DOMAINID</description>

       <config-property-name>localAccessPointSpec</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

      </config-property>

      <config-property>

        <description>factory-wise SSL to configure whether mutual
authentication is required, default to false</description>

<config-property-name>mutualAuthenticationRequired</config-property-name>

        <config-property-type>java.lang.Boolean</config-property-type>
```

```
        </config-property>

        <config-property>

          <description>factory-wise SSL for configuring identity key store
file name, must be configured if SSL is desired</description>

<config-property-name>identityKeyStoreFileName</config-property-name>

          <config-property-type>java.lang.String</config-property-type>

        </config-property>

        <config-property>

          <description>factory-wise SSL setting for private key alias used
in the key store, must be configured if SSL is desired</description>

          <config-property-name>privateKeyAlias</config-property-name>

          <config-property-type>java.lang.String</config-property-type>

        </config-property>

        <config-property>

          <description>factory-wise trusted key store file name, must be
configured if SSL is desired</description>

<config-property-name>trustedKeyStoreFileName</config-property-name>

          <config-property-type>java.lang.String</config-property-type>

        </config-property>

        <config-property>

          <description>factory-wise password for identityKeyStore in clear
text</description>

<config-property-name>identityKeyStorePassPhrase</config-property-name>

          <config-property-type>java.lang.String</config-property-type>

        </config-property>

        <config-property>
```

```
        <description>factory-wise password for privateKeyAlias in clear
text</description>

<config-property-name>privateKeyAliasPassPhrase</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>

    <config-property>

        <description>factory-wise password for trustedKeyStore in clear
text</description>

<config-property-name>trustedKeyStorePassPhrase</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>

    <config-property>

     <description>factory-wise RemoteAccessPoint specification of the
format //hostname:port/domainId=DOMAINID</description>

        <config-property-name>remoteAccessPointSpec</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>

    <config-property>

    <description>factory-wise allow anonymous access to Tuxedo, default
to false</description>

        <config-property-name>rapAllowAnonymous</config-property-name>

        <config-property-type>java.lang.Boolean</config-property-type>

    </config-property>

    <config-property>

    <description>factory-wise application key value for anonymous user,
default to -1</description>

<config-property-name>rapDefaultApplicationKey</config-property-name>
```

```
            <config-property-type>java.lang.Integer</config-property-type>

        </config-property>

        <config-property>

            <description>factory-wise application key fully qualified class
name for AppKey generator</description>


<config-property-name>rapApplicationKeyClass</config-property-name>

            <config-property-type>java.lang.String</config-property-type>

        </config-property>

        <config-property>

            <description>factory-wise custom application key
parameter</description>


<config-property-name>rapApplicationKeyClassParam</config-property-name>

            <config-property-type>java.lang.String</config-property-type>

        </config-property>

        <config-property>

            <description>factory-wise session profile block timeout value,
default to 60000 milliseconds</description>

            <config-property-name>spBlockTime</config-property-name>

            <config-property-type>java.lang.Integer</config-property-type>

        </config-property>

        <config-property>

            <description>factory-wise whether allows interoperate with 6.5
Tuxedo Domain, default to false</description>

            <config-property-name>spInteroperate</config-property-name>

            <config-property-type>java.lang.Boolean</config-property-type>

        </config-property>

        <config-property>
```

```
        <description>factory-wise security setting, legal values: NONE,
DM_PW, APP_PW</description>

        <config-property-name>spSecurity</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>

    <config-property>

        <description>factory-wise credential propagation policy, either
LOCAL or GLOBAL</description>

        <config-property-name>spCredentialPolicy</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>

    <config-property>

        <description>factory-wise number of seconds that session waits
between automatic connection establishment,

                    default to 60 seconds. A value of 0 disabled connection
retry</description>

        <config-property-name>spRetryInterval</config-property-name>

        <config-property-type>java.lang.Long</config-property-type>

    </config-property>

    <config-property>

        <description>factory-wise maximum number of times adapter will try
to establish a session connection to

                    remote Tuxedo access point. Default value is
Long.MAX_VALUE.</description>

        <config-property-name>spMaxRetries</config-property-name>

        <config-property-type>java.lang.Long</config-property-type>

    </config-property>

    <config-property>
```

```
        <description>factory-wise compression threshold, default to
Integer.MAX_VALUE</description>

        <config-property-name>spCompressionLimit</config-property-name>

        <config-property-type>java.lang.Integer</config-property-type>

      </config-property>

      <config-property>

        <description>factory-wise minimum encryption strength requirement,
legal values are 0, 40, 56, 128, 256.

                        Default value is 0.</description>

        <config-property-name>spMinEncryptBits</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

      </config-property>

      <config-property>

        <description>factory-wise maximum encryption strength requirement,
legal values are 0, 40, 56, 128, 256.

                        Default value is 128.</description>

        <config-property-name>spMaxEncryptBits</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

      </config-property>

      <config-property>

        <description>factory-wise the maximum idle time before sending
application level keep alive.

                      It is measured in millisecond, and roundup to seconds.
Default value is 0.</description>

        <config-property-name>spKeepAlive</config-property-name>

        <config-property-type>java.lang.Long</config-property-type>

      </config-property>

      <config-property>
```

```
        <description>factory-wise how long adapter will wait for
acknowledgement before adapter decides the

                    connection already lost.  Measurement in millisecond,
and its default value is 10 seconds.

                    A value of 0 will disable the wait, and thus will not
close the connection</description>

        <config-property-name>spKeepAliveWait</config-property-name>

        <config-property-type>java.lang.Long</config-property-type>

    </config-property>

    <config-property>

        <description>factory-wise valid Tuxedo service names in a
comma-separated list.  If not specified then

                    default import will be used and will grant all service
requests to remote Tuxedo domain</description>

        <config-property-name>impResourceName</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>

    <config-property>

        <description>Exported resources.  Types of resource supported
are</description>

        <config-property-name>exportSpec</config-property-name>

        <config-property-type>java.lang.String</config-property-type>

    </config-property>


<connectionfactory-interface>javax.resource.cci.ConnectionFactory</connect
ionfactory-interface>


<connectionfactory-impl-class>com.oracle.tuxedo.adapter.cci.TuxedoConnecti
onFactory</connectionfactory-impl-class>
```

```
<connection-interface>javax.resource.cci.Connection</connection-interface>

<connection-impl-class>com.oracle.tuxedo.adapter.cci.TuxedoJCAConnection</
connection-impl-class>
      </connection-definition>
<!--
      Other types of transaction support
      <transaction-support>NoTransaction</transaction-support>
      <transaction-support>LocalTransaction</transaction-support>
-->
      <transaction-support>XATransaction</transaction-support>
      <authentication-mechanism>

<authentication-mechanism-type>BasicPassword</authentication-mechanism-typ
e>

<credential-interface>javax.resource.spi.security.PasswordCredential</cred
ential-interface>
      </authentication-mechanism>
      <reauthentication-support>false</reauthentication-support>
    </outbound-resourceadapter>
  </resourceadapter>
</connector>
```

WebLogic Server Specific Side File

To use the Tuxedo JCA Adapter running in WebLogic Server, you must configure the JNDI name to a file with the specific name "weblogic-ra.xml". This file should be put in the META-INF of the Resource Archive as shown in Listing 59.

Listing 59   WebLogic Server Specific Side File

```xml
<?xml version="1.0"?>
<weblogic-connector
 xmlns="http://www.bea.com/ns/weblogic/90">
  <jndi-name>eis/TuxedoConnector</jndi-name>
  <enable-access-outside-app>true</enable-access-outside-app>
  <enable-global-access-to-classes>true</enable-global-access-to-classes>
  <outbound-resource-adapter>
    <connection-definition-group>

<connection-factory-interface>javax.resource.cci.ConnectionFactory</connection-factory-interface>
      <connection-instance>
        <jndi-name>eis/TuxedoConnectionFactory</jndi-name>
      </connection-instance>
    </connection-definition-group>
  </outbound-resource-adapter>
</weblogic-connector>
```

Using the JDeveloper SOA Suite

The Tuxedo JCA Adapter has an SOA configuration wizard in a separate `.jar` file that supports JDeveloper SOA Extension. By using this graphical IDE, it allows an SOA application to leverage Oracle Tuxedo hosted services through the Tuxedo JCA Adapter. You can create Oracle Tuxedo"External References" by dragging-and-dropping the Tuxedo JCA Adapter icon and configure its SOA operations using this configuration wizard.

There is no change to either JATMI or CCI interfaces; however, the JDeveloper SOA Extension can only use JCA standard CCI interface. Although Tuxedo JCA Adapter does support inbound

service request but currently the Tuxedo JCA Adapter SOA configuration wizard only support outbound SOA operation.

JDeveloper Studio IDE

The Tuxedo JCA Adapter SOA configuration wizard is available to users who choose to use JDeveloper with an SOA Extension as their development environment when designing business logic that uses an Oracle Tuxedo application through the Tuxedo JCA Adapter. You must install SOA Extension for JDeveloper.

The Tuxedo JCA Adapter SOA configuration wizard helps you to navigate through the configuration pages. These pages include the Welcome Page, Service Name page, JNDI connection factory page, operation page, and Tuxedo JCA Adapter properties page. The Tuxedo JCA Adapter properties page contains `com.oracle.tuxedo.adapter.cci.TuxedoInteractionSpec` properties.

Oracle Tuxedo JCA Adapter Configuration

Even though the name of the component contains the name "Configuration wizard", this is a wizard that configures an SOA Extension in JDeveloper so that SOA knows how to invoke the Tuxedo JCA Adapter.It is required that you must configure the Tuxedo JCA Adapter.   There is no special configuration required for you to configure a Tuxedo JCA Adapter that works with SOA framework in runtime.

An existing Tuxedo JCA Adapter installation continues to work without modification to the configuration. For a new application, you must choose from one of the three configuration styles to configure the Tuxedo JCA Adapter.

SOA Extension Configuration

Listing 60 shows the new Tuxedo JCA Adapter type to be configure in the SOA configuration file, `soa-config.xml`.

Listing 60   soa-config.xml

```
<adapterType
resourceBundle="com.oracle.tuxedo.adapter.designtime.resource.TuxedoJcaAda
pterResourceBundle">

  <name>${TJA_ADAPTER_COMPONENT_NAME_L}</name>
```

```
    <bindingType>jca</bindingType>

     <bindingSubType> tuxedo</bindingSubType>

     <createType>referenceOnly</createType>


<implementationClass>com.oracle.tuxedo.adapter.designtime.TjaScaEndpointIm
pl</implementationClass>

          <description>${TJA_ADAPTER_COMPONENT_DESC}</description>

          <tooltip>${TJA_ADAPTER_COMPONENT_DESC}</tooltip>


<icon16x16>/com/oracle/tuxedo/adapter/designtime/resource/tja_adapter_16x1
6.png</icon16x16>
<icon20x20>/com/oracle/tuxedo/adapter/designtime/resource/tja_adapter_20x2
0.png</icon20x20>
<topSectionIcon>oracle/tip/tools/ide/fabric/resource/image/visuals_rd1/whi
teServiceTop.png</topSectionIcon>
<middleSectionIcon>oracle/tip/tools/ide/fabric/resource/image/visuals_rd1/
whiteServiceMiddle.png</middleSectionIcon>
<bottomSectionIcon>oracle/tip/tools/ide/fabric/resource/image/visuals_rd1/
whiteServiceBottom.png</bottomSectionIcon>
<collapsedSectionIcon>oracle/tip/tools/ide/fabric/resource/image/visuals_r
d1/whiteServiceCollapsed.png</collapsedSectionIcon>

</adapterType>
```

The above configuration is actually available as part of the download;you do not really need to modify it.

The SOA configuration file is located in the
`$JDEVELOPER_HOME/integration/seed/soa/configuration` directory.

Starting JDeveloper with Oracle Tuxedo JCA Adapter SOA Configuration Wizard

You can start the Oracle JDeveloper IDE by either double clicking the JDeveloper icon, or by starting it using `$JDEVELOPER_HOME/jdev/bin/jdev`. By scrolling down the slider in the

Component palette, you can find an "Oracle Tuxedo" icon for the Tuxedo JCA Adapter. Figure 1provides a screenshot of JDeveloper with Tuxedo JCA Adapter enabled through SOA Extension.

Figure 1   Configuring Using Tuxedo JCA Adapter SOA
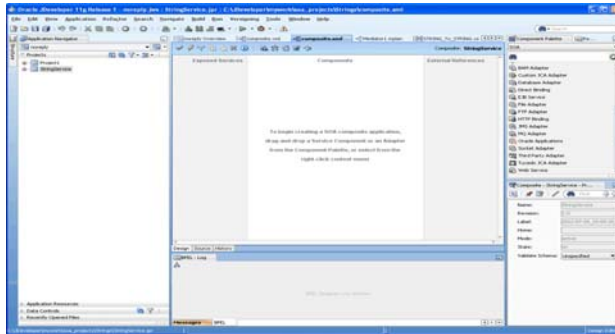
Configuration Wizard

The following are step-by-step procedures to configure an "External Reference" to the Oracle Tuxedo service TOUPPER that can be invoked by SOA framework at runtime to access an Oracle Tuxedo service through the Tuxedo JCA Adapter.

Create Tuxedo JCA Adapter External References

Welcome Page

Scroll down the "Component Palette" from panel right-hand side to find "Tuxedo JCA Adapter", you can drag-and-drop the "Tuxedo JCA Adapter'" icon to the "External References" area in the middle window pane.The Tuxedo JCA Adapter SOA configuration wizard"Welcome" page appears as shown in Figure 2.

Figure 2   Welcome Page

Service Name Page

Click the "Next" button on the "Welcome" page and the Service Name page is displayed as shown in Figure 3. You can enter the same service name as the Oracle Tuxedo exported service name, or any name that makes sense for an SOA application. This Service Name is for SOA reference.

Figure 3   Service Name Page
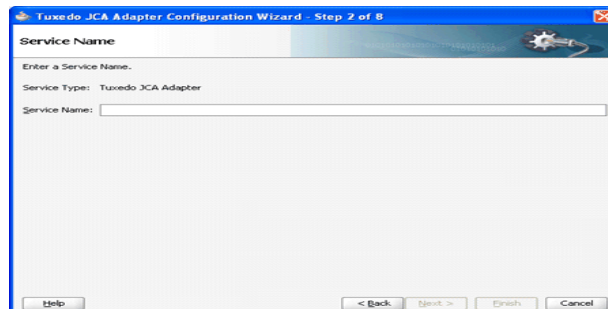
In this example we'll enter "STRING_TOUPPER" as shown in Figure 4

Figure 4 STRING_TOUPPER
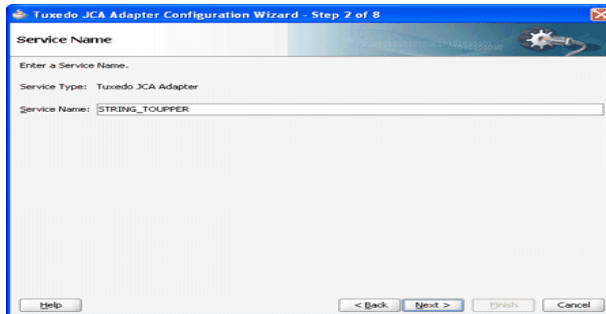
Connection Factory Information Page

Click the "NEXT" button on "Service Name" page.The "Connection Factory Information" page appears as shown in Figure 5. The "Connection Factory Information" page allows you to specify which Tuxedo JCA Adapter connection factory to use by specifying the JNDI name of the intended connection factory.

Figure 5 Connection Factory Information Page
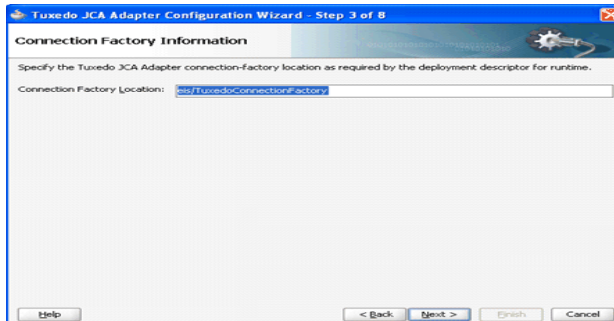
Enter the JNDI name for the connection factory you intend to use. In this example we'll use the default JNDI name "eis/TuxedoConnectionFactory".

Adapter Interface Page

Click the "NEXT" button. The Adapter Interface Page appears as shown inFigure 6. The "Adapter Information" page is used to construct WSDL and specifies the XSD schema for the data buffer.

Figure 6   Adapter Interface Page
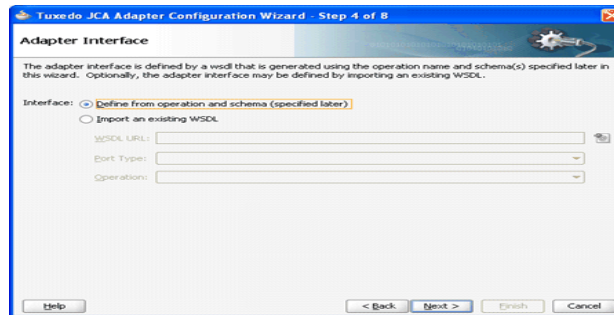
In most situations, it is sufficient for to select the "Define from operation and schema (specified later)" button; however, you can also import an existing WSDL.

Operation Page

Click the "NEXT" button and the "Operation" page appears as shown in Figure 7. Leave the "Outbound" value alone as the Tuxedo JCA Adapter SOA configuration wizard only supports "Outbound Operation".

Figure 7   Operation Page
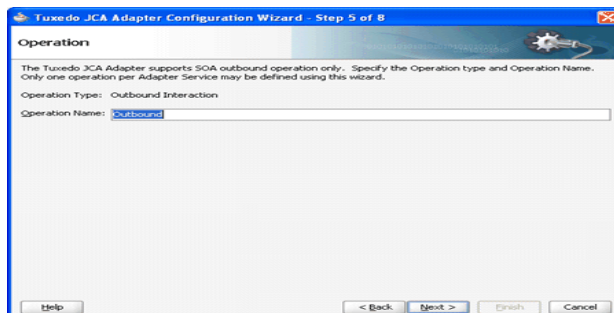
Oracle Tuxedo Interaction Properties Page

Click the "NEXT" button and the "Oracle Tuxedo Interaction Properties" page appears as shown in Figure 8.   You must enter the actual Tuxedo JCA Adapter imported Tuxedo service name.

Figure 8   Oracle Tuxedo Interaction Properties Page
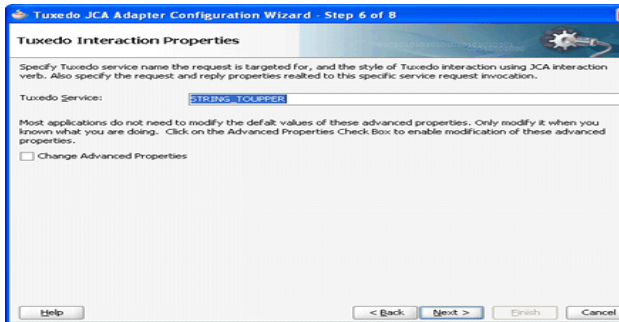
In this example, the imported Tuxedo service name is "STR_TOUPPER". Enter "STR_TOUPPER" as the Oracle Tuxedo service as shown in Figure 9.

Figure 9   STR_TOUPPER
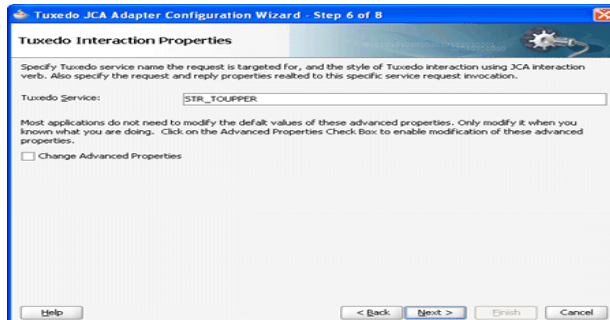
Tuxedo Request Buffer Properties

Click the "NEXT" button and the "Oracle Tuxedo Request Buffer Properties" page appears as shown in Figure 10. Specify the Tuxedo JCA Adapter Interaction Request buffer type for the targeted Oracle Tuxedo service.

If the Oracle Tuxedo buffer type is one of the structure buffer types (such as FML/FML32/VIEW/VIEW32/X_C_TYPE/X_COMMON), you must provide a corresponding Java class generated from the Field Table or View Table Definition file. These class can be generated using tools provided by Tuxedo JCA Adapter, and they must match the user-defined XSD d for that buffer.

Figure 10   Tuxedo Request Buffer Properties
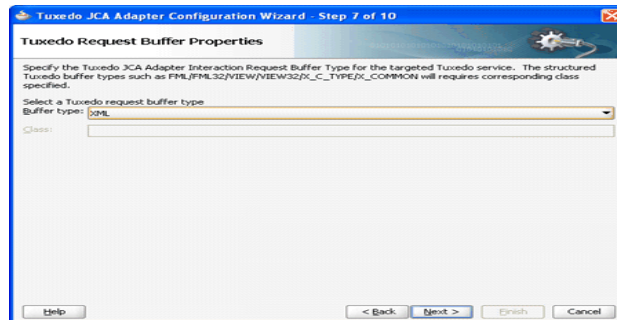
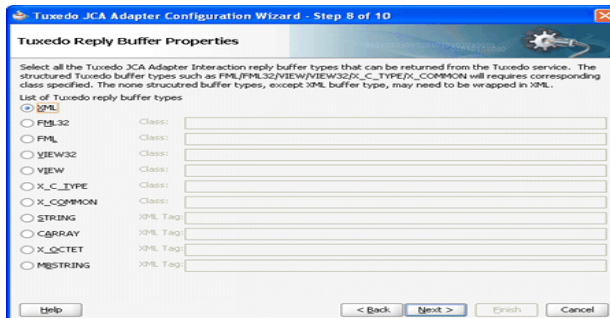In this example we use "STRING" buffer type selected from the drop down menu as shown in Figure 11.

Figure 11   STRING Buffer Type
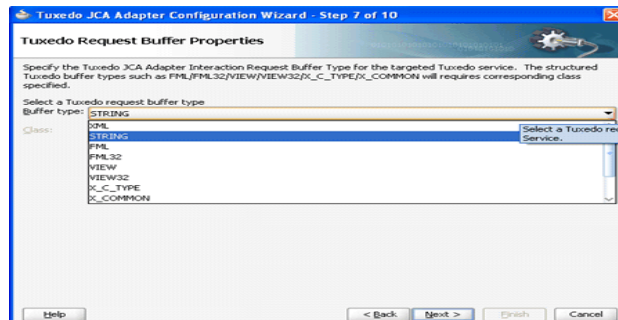

Oracle Tuxedo Reply Buffer Properties Page

Click the "NEXT" button and the "Oracle Tuxedo Reply Buffer Properties" appears as shown in Figure 12. Select all the Tuxedo JCA Adapter Interaction reply buffer types that can be returned from an Oracle Tuxedo Service.

Figure 12   Tuxedo Reply Buffer Properties Page

In this example, selects "STRING" as reply buffer type as shown in Figure 13.

Figure 13   String Buffer
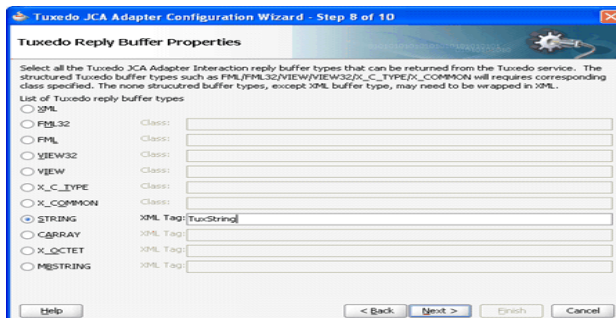
Enter the XML tag according to the schema you want to use.

Message Page

Click on "NEXT" button and the "Message" page appears as shown in Figure 14. The "Message" configuration is used to configure the corresponding schema. You must specify the schema file location and select the schema element that defines the outbound message.

Depends on whether a reply is required or not the "Message" page can be different.

When Reply Is Required

Figure 14 shows the Message configuration wizard page when a reply is required. Notice there are two schema input fields; the first one is labeled "Inbound" which is the schema required for
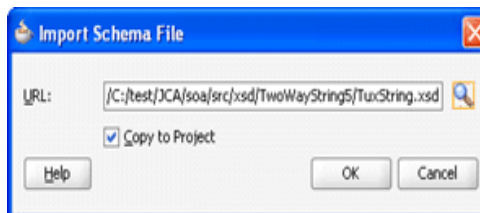
request message; the second one is labeled "Outbound" which is the schema required for reply message.

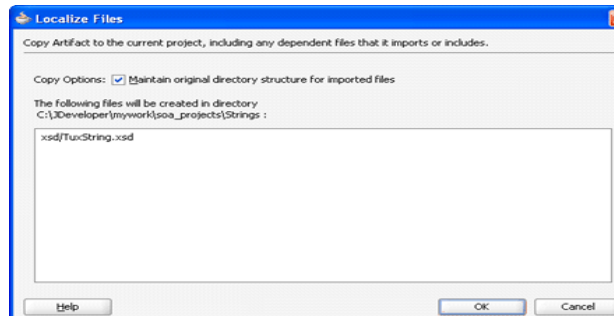**Figure 14   Message Configuration Page**
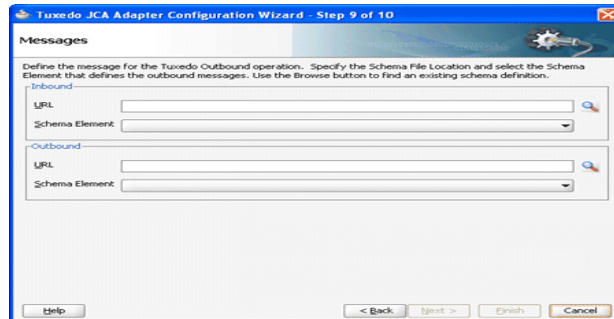
Click on the search icon" to import the schema file as shown in Figure 15.

Figure 15   Import Schema File

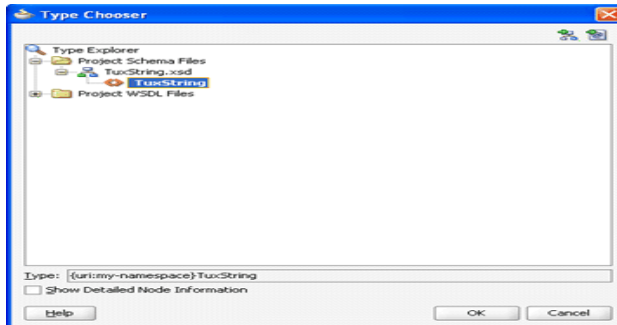

Click "OK" and the Localize Files page appears as shown in Figure 16.
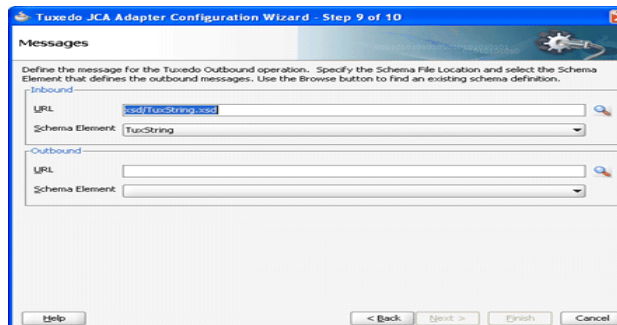
Figure 16   Localize Files

Highlight the element for the String buffer as shown in Figure 17

Figure 17 Type Chooser

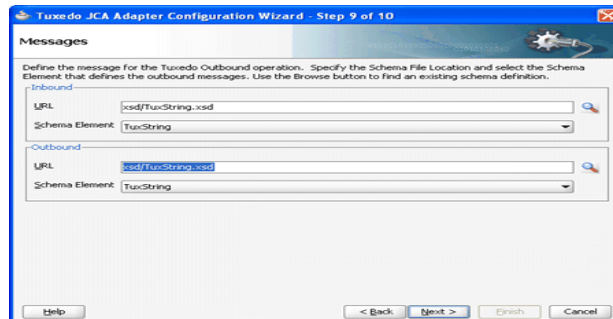

Click "OK" and the "Inbound" part of the schema appears as shown in Figure 18.

Figure 18 Inbound

Do the same for Outbound Request, the "Message" page will look like Figure 19.

Figure 19 Outbound

When Reply Is Not Required

Figure 20 shows the Message configuration wizard page when reply is not required. (by selecting TPNOREPLY in the "Tuxedo Interaction Properties" page). Notice there is only one schema input field. It is labeled "Message Schema" which is the schema required for request message
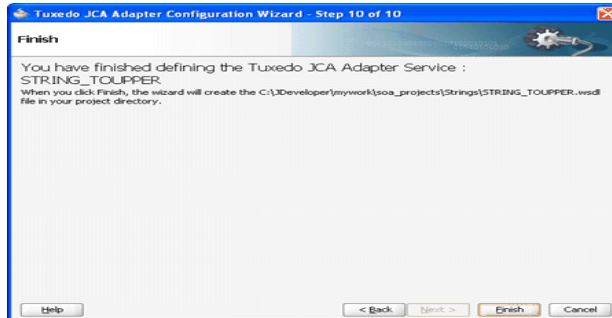
Figure 20 Message Schema
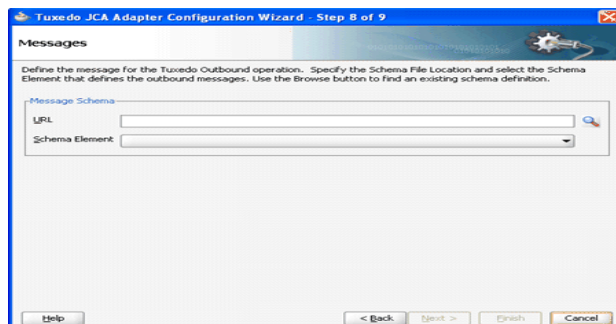
Finish Page

Click the "NEXT" button and the "Finish" page appears as shown in Figure 21.

Figure 21   Finish Page

Click the "Finish" button and an Tuxedo JCA Adapter external reference is created as shown in Figure 22.

Figure 22   External reference
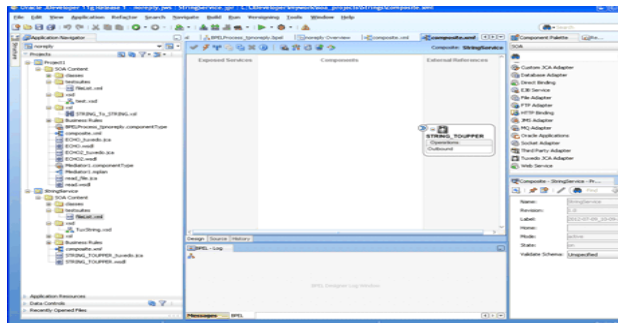
Buffer Type XSD Requirements

FML Buffer Type XSD Requirements

The corresponding class is generated using an Oracle Tuxedo FML table as input for the `weblogic.wtc.jatmi.mkfldclass` compiler to generate the class. The XSD must indicate that the XML is enclosed in an element with tag name "FML" as shown in Listing 81.

Listing 81   XML is enclosed in an element with tag name "FML"

```
<ns:FML xmlns:ns="uri:my-namespace">

  <TEST_SHORT>1234</TEST_SHORT>
```

```
   <TEST_STRING>hello</TEST_STRING>

</ns:FML>
```

The corresponding schema FML is shown in Listing 82.

Listing 82   Output FML Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="FML">

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="TEST_SHORT" type="xsd:string"/>

        <xsd:element name="TEST_STRING" type="xsd:string"/>

      </xsd:sequence>

    </xsd:complexType>

  </xsd:element>

</xsd:schema>
```

FML32 Buffer Type XSD Requirements

The corresponding class is generated using a Tuxedo FML32 table as input for the `weblogic.wtc.jatmi.mkfldclass32` compiler to generate the class. The XSD must indicate that the XML is enclosed in an element with tag name "FML32" as shown in Listing 83.

Listing 83   XML Enclosed Element Tag "FML32"

```
<ns:FML32 xmlns:ns="uri:my-namespace">

  <TEST_SHORT>1234</TEST_SHORT>

  <TEST_STRING>hello</TEST_STRING>

</ns:FML32>
```

The corresponding schema FML is shown in Listing 84.

Listing 84   Output FML32 Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="FML32">

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="TEST_SHORT" type="xsd:string"/>

        <xsd:element name="TEST_STRING" type="xsd:string"/>

      </xsd:sequence>

    </xsd:complexType>

  </xsd:element>

</xsd:schema>
```

VIEW Buffer Type XSD Requirements

The corresponding class is generated using an Oracle Tuxedo VIEW definition file as input for the `weblogic.wtc.jatmi.viewj` compiler to generate the class. The XSD must indicate that the XML is enclosed in an element tag named "`VIEW`". The buffer sub-type is the name defined in VIEW definition file and should be used as XML tag name that encloses the actual payload as shown Listing 85.

Listing 85   XML Enclosed in Element Tag "VIEW"

```
<ns:VIEW xmlns:ns="uri:my-namespace">

  <MyView>

    <TEST_SHORT>1234</TEST_SHORT>

    <TEST_STRING>hello</TEST_STRING>

  </MyView>

</ns:VIEW>
```

The "MyView" tag is the VIEW buffer sub-type.

The corresponding schema VIEW is shown in Listing 86.

Listing 86   Output View Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="VIEW">

    <xsd:complexType>
```

```
    <xsd:sequence>

      <xsd:element name="View16">

        <xsd:complexType>

          <xsd:sequence>

            <xsd:element name="TEST_SHORT" type="xsd:string"/>

            <xsd:element name="TEST_STRING" type="xsd:short"/>

          </xsd:sequence>

        </xsd:complexType>

      </xsd:element>

    </xsd:sequence>

  </xsd:complexType>

</xsd:element>

</xsd:schema>
```

VIEW32 Buffer Type XSD Requirements

The corresponding class is generated using an Oracle Tuxedo VIEW32 definition file as input for the `weblogic.wtc.jatmi.viewj32` compiler to generate the class. The XSD must indicate that the XML is enclosed in an element tag named "VIEW32". The buffer sub-type is the name defined in VIEW definition file and should be used as an XML tag name that encloses the actual payload as shown in Listing 87.

Listing 87   XML is enclosed in an element with tag name "VIEW32"

```
<ns:VIEW32 xmlns:ns="uri:my-namespace">

  <View32>

    <TEST_SHORT>1234</TEST_SHORT>

    <TEST_STRING>hello</TEST_STRING>

  </view32>

</ns:VIEW32>
```

The "View32" tag is the buffer sub-type.

The corresponding schema VIEW32 is shown in Listing 88.

Listing 88   Output View32 Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="VIEW32">

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="View32">

          <xsd:complexType>

            <xsd:sequence>

              <xsd:element name="TEST_SHORT" type="xsd:string"/>

              <xsd:element name="TEST_STRING" type="xsd:short"/>

            </xsd:sequence>

          </xsd:complexType>

        </xsd:element>

      </xsd:sequence>

    </xsd:complexType>

  </xsd:element>

</xsd:schema
```

X_C_TYPE Buffer Type Requirements

The corresponding class is generated using an Oracle Tuxedo VIEW definition file as input for the `weblogic.wtc.jatmi.viewj` compiler with command line option "`-xctype`" to generate the class. The XSD must indicate that the XML is enclosed in an element tag named "`X_C_TYPE`". The buffer sub-type is the name defined in VIEW definition file and should be used as a XML tag name that encloses the actual payload as shown in Listing 89.

Listing 89 ML is enclosed in an element with tag name "X_C_TYPE"

```
<ns:X_C_TYPE xmlns:ns="uri:my-namespace">

  <MyXCType>

    <TEST_SHORT>1234</TEST_SHORT>

    <TEST_STRING>hello</TEST_STRING>

  </MyXCType>

</ns:X_C_TYPE>
```

The "`MyXCType`" tag is the buffer sub-type.

The corresponding schema X_C_TYPE is shown in Listing 88.

Listing 90 Output X_C_TYPE Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<xsd:element name="X_C_TYPE">

  <xsd:complexType>

    <xsd:sequence>

      <xsd:element name="MyXCType">

        <xsd:complexType>

          <xsd:sequence>

            <xsd:element name="TEST_SHORT" type="xsd:string"/>

            <xsd:element name="TEST_STRING" type="xsd:short"/>

          </xsd:sequence>

        </xsd:complexType>

      </xsd:element>

    </xsd:sequence>

  </xsd:complexType>

</xsd:element>

</xsd:schema>
```

X_COMMON Buffer Type XSD Requirement

The corresponding class is generated using an Oracle Tuxedo VIEW definition file as input for the `weblogic.wtc.jatmi.viewj` compiler with command line option "-xcommon" to generate the class. The XSD must indicate that the XML is enclosed in an element tag named "X_COMMON". The buffer sub-type is the name defined in VIEW definition file and should be used as a XML tag name that encloses the actual payload as shown in Listing 91.

Listing 91 XML is enclosed in an element with tag name "X_COMMON"

```
<ns:X_COMMON xmlns:ns="uri:my-namespace">

  <MyXCommon>

    <TEST_SHORT>1234</TEST_SHORT>

    <TEST_STRING>hello</TEST_STRING>
```

```
        </MyXCommon>

</ns:X_COMMON>
```

The "`MyXCommon`" tag is the X_COMMON buffer sub-type.

The corresponding schema X_C_TYPE is shown in Listing 92.

Listing 92   Output X_C_TYPE Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="X_COMMON">

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="MyXCommon">

          <xsd:complexType>

            <xsd:sequence>

              <xsd:element name="TEST_SHORT" type="xsd:string"/>

              <xsd:element name="TEST_STRING" type="xsd:string"/>

            </xsd:sequence>

          </xsd:complexType>

        </xsd:element>

      </xsd:sequence>

    </xsd:complexType>
```

```
    </xsd:element>

</xsd:schema>
```

WSDL and Interaction Verb Selection

Since SOA is universally WSDL-based, the WSDL operation signature decides which "verb" to use. SYNC_SEND_RECEIVE corresponds to WSDL as shown in Listing 92.

Listing 93   SYNC_SEND_RECEIVE corresponds to WSDL

```
<operation name="ECHO">

  <input message:="tns:send_msg"/>

  <output message="tns:recv_msg"/>

</operation>
```

SYNC_SEND corresponds to WSDL as shown in Listing 94.

Listing 94   SYNC_SEND

```
<operation name="MYEVENT">

  <input message:="tns:write_msg"/>

</operation>
```

"SYNC_RECEIVE" is not supported by WSDL operation.

Buffer Transformation

The SOA framework works with the XML record Oracle Tuxedo block oriented buffer types have to be converted into an XML record, and vice-versa. The Tuxedo JCA Adapter performs buffer transformation under the cover according to rules described in this section and with the

help of the user-configured metadata configured. No coding required by the client or SOA Frameworkt.

Input Data Conversion

The input data means the XML record received from SOA Framework by the Tuxedo JCA Adapter for service request targeted to Oracle Tuxedo. Table 31 gives a matrix for input data conversion from SOA Framework to Oracle Tuxedo.

Table 31 Input Buffer Type Conversion Table

Input Buffer Type Conversion Table		
Input SOA Record Type	Tuxedo Buffer Type Configured	Conversion Type
XML Record	XML	No Conversion
XML Record	FML & FML32	Type 1
XML Record	VIEW & VIEW32	Type 2
XML Record	STRING	Type 3
XML Record	CARRAY, X_OCTET	Type 3
XML Record	X_C_TYPE, X_COMMON	Type 2
XML Record	MBSTRING	Type 3

If you did not specify an Oracle Tuxedo service input buffer type, the input XML record is converted before forwarding the request to Oracle Tuxedo in an XML buffer.

If an Oracle Tuxedo service supports multiple Oracle Tuxedo buffer types and the corresponding SOA composite application can generate requests with more than one type of an Oracle Tuxedo buffer, you should configure the same service multiple times using configuration wizard. This causes more than one `TuxedoInteractionSpec` to be available to SOA framework invoking an Oracle Tuxedo service.

If the targeted Oracle Tuxedo service requires FML/FML32 as an input buffer type, a request Class must be configured with a corresponding FML/FML32 class. If not specified, the configuration wizard will not allow you to advance to the next wizard page. If the actual received

input XML record contains a tag that there is no corresponding FML/FML32 field name, then a `ResourceException` is thrown when SOA Framework invokes the Tuxedo JCA Adapter.

Note: The same requirement also applies to VIEW, VIEW32, X_C_TYPE, and X_COMMON.

Output Data Conversion

Output data means the user data in the reply returned by an Oracle Tuxedo service. In this case, the Tuxedo JCA Adapter can only receive data in an Oracle Tuxedo typed buffer. The data is converted to an XML record with minimum user-configured metadata.

An Oracle Tuxedo service potentially can return user data using different Oracle Tuxedo typed buffers. In this case, the Tuxedo JCA Adapter implicitly converts it to an XML record; no client code needs to be written. However, you are responsible for configuring the required metadata.

Table 32 provides a matrix for output data conversion from an Oracle Tuxedo buffer to an XML record.

Table 32 Output Buffer Type Conversion Table

Output Buffer Type Conversion Table		
Input Tuxedo Buffer Type	Output SOA Record Type	Conversion Type
XML	XML Record	No Conversion
FML & FML32	XML Record	Type 1
VIEW & VIEW32	XML Record	Type 2
STRING	XML Record	Type 3
CARRAY_X_OCTET	XML Record	Type 3
X_C_TYPE & X_COMMON	XML Record	Type 2
MBSTRING	XML Record	Type 3

If the reply buffer type is one of VIEW/VIEW32/X_C_TYPE/X_COMMON and you did not configure the corresponding property for the class name `TuxedoInteractionSpec.`, a `ResourceException` exception is thrown. The same requirement also applies to FML/FML32 buffer types.

If the reply buffer type is one of STRING/CARRAY/X_OCTET/MBSTRING and you did not configure an appropriate XML tag, a `ResourceException` exception is thrown.

Conversion Types

The following sections explain how the Tuxedo JCA Adapter handles the buffer type conversion.

No Conversion

"No Conversion" means the Tuxedo JCA Adapter returns a reply buffer, or accepts a request with direct mapping. If the reply XML buffer does not contain a well-formed XML document, a `ResourceException` is thrown.

Type 1 Conversion

"Type 1" conversion means the Tuxedo JCA Adapter uses FLDID to retrieve FIELD NAME. It uses an XML element tag for both request and reply as handled by ALSB Tuxedo Transport. In this case, you must configure an FML class name for FML and an FML32 class name for FML32 as shown in Table 33.

Table 33 XML Root Element Tag Name

XML Root Element Tag Name	
Tuxedo Buffer Type	**XML Root Element Tag**
FML	<FML>
FML32	<FML32>

Embedded FML32 is supported by the Tuxedo JCA Adapter as well. However, the field type FLD_MBSTRING is not supported in the FML/FML32 to XML conversion.

Listing 95shows is an FML32 table example.

Listing 95   FML32 Table Example.

```
#name               number type    flags  comments

TEST_SHORT 108      short   -       -

TEST_STRING 109     string  -       -
```

Using the above FML32 table as input for `weblogic.wtc.jatmi.mkfldclass32` with package name `tuxedo.test.datarecord.FML32` creates a Java class as shown in Listing 96

Listing 96   Java Class Example

```
package tuxedo.test.datarecord.FML32;

import java.io.*;

import java.lang.*;

import java.util.*;

import weblogic.wtc.jatmi.*;


public final class fmltbl32

        implements weblogic.wtc.jatmi.FldTbl

{

  Hashtable nametofieldHashTable;

  Hashtable fieldtonameHashTable;

  /** number: 108  type: short */

  public final static int TEST_SHORT = 108;

  /** number: 109  type: string */

  public final static int TEST_STRING = 167772269;


  public String Fldid_to_name(int fldid)

  {

    if ( fieldtonameHashTable == null ) {

      fieldtonameHashTable = new Hashtable();

      fieldtonameHashTable.put(new Integer(TEST_SHORT), "TEST_SHORT");
```

```
      fieldtonameHashTable.put(new Integer(TEST_STRING), "TEST_STRING");

    }


    return ((String)fieldtonameHashTable.get(new Integer(fldid)));

  }


  public int name_to_Fldid(String name)

  {

    if ( nametofieldHashTable == null ) {

      nametofieldHashTable = new Hashtable();

      nametofieldHashTable.put("TEST_SHORT", new Integer(TEST_SHORT));

      nametofieldHashTable.put("TEST_STRING", new Integer(TEST_STRING));

    }


    Integer fld = (Integer)nametofieldHashTable.get(name);

    if (fld == null) {

      return (-1);

    } else {

      return (fld.intValue());

    }

  }


  public String[] getFldNames()

  {

    String retval[] = new String[2];

    retval[0] = new String("TEST_SHORT");

    retval[1] = new String("TEST_STRING");
```

```
        return retval;

    }

}
```

Listing 97 shows what it looks like after it is converted from FML32 buffer.

Listing 97   XML Document Converted from an FML32 Buffer

```
<ns:FML32 xmlns:ns="uri:my-namespace">

  <TEST_SHORT>1234</TEST_SHORT>

  <TEST_STRING>hello</TEST_STRING>

</ns:FML32>
```

Listing 98 shows the corresponding Schema:

Listing 98   Corresponding Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="FML32">

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="TEST_SHORT" type="xsd:string"/>
```

```
        <xsd:element name="TEST_STRING" type="xsd:string"/>

    </xsd:sequence>

  </xsd:complexType>

</xsd:element>

</xsd:schema>
```

Note: The schema shows the FML32 with embedded VIEW32 and FML32.

Type 2 Conversion

When you select VIEW/VIEW32/X_C_TYPE/X_COMMON as request and/or reply buffer type as shown in Listing 34, a fully qualified Java class corresponding to the Oracle Tuxedo buffer type is needed. You can enter Java class information in a Text Input Field.

X_C_TYPE and X_COMMON are similar to VIEW and thus they can be treated/created the same way as VIEW.

The fully qualified class path will be part of the expanded `TuxedoInteractionSpec`.

Table 34 XML Root Element Tag Name

XML Root Element Tag Name	
Tuxedo Buffer Type	**XML Root Element Tag**
VIEW	<VIEW>
VIEW32	<VIEW32>
X_C_TYPE	<X_C_TYPE>
X_COMMON	<X_COMMON>

You must specify an element of the same name as sub-type as the wrapping XML wrapping element in the corresponding schema file. For instance, if the sub-type name is "`myview32`" then the corresponding XML document should look like Listing 99

Listing 99   myview32 XML Document

```
<VIEW32>

  <myview32>

 …

  </myview32>

</VIEW32>
```

And the XSD file for this VIEW32 buffer type can be defined as shown in Listing 100.

Listing 100   View32 XSD File

```
<xsd:element name="VIEW32">

  <xsd:complexType>

    <xsd:sequence>

      <xsd:element name="myview32">

        <xsd:complexType>

        …

        </xsd:complexType>

      </xsd:element>

    </xsd:sequence>

  </xsd:complexType>

</xsd:element>
```

Nested VIEW currently is not supported by the Tuxedo JCA Adapter.

Listing 101 shows a VIEW definition file that can be used with
`weblogic.wtc.jatmi.viewj32`.

Listing 101   VIEW Definition File

```
VIEW View32
short  TEST_SHORT -  1  -  -  0
string TEST_STRING  -  1  -  100   -
END
```

Listing 103shows VIEW32 class file generated using viewj32 compiler.

Listing 102   viewj32 Compiler Generated View32 Class File

```
package tuxedo.test.datarecord.VIEW32;
import java.io.*;
import java.lang.*;
import weblogic.wtc.jatmi.*;
import com.bea.core.jatmi.common.Utilities;

public class View32 extends TypedView32 implements Serializable {
  private short TEST_SHORT = 0;
  private String TEST_STRING = null;
  private boolean _associatedFieldHandling = false;

  public View32() {
    super("View32");
    return;
  }
  /**
   * Gets the current state of associated field handling.
```

```
 * @return the current state (true=on, false=off)

 */

public boolean getAssociatedFieldHandling()

{

  return _associatedFieldHandling;

}

/**

 * Sets the state of associated field handling.

 * @param state the desired state (true=on, false=off)

 */

public void setAssociatedFieldHandling(boolean state)

{

  _associatedFieldHandling = state;

}


/**

 * Gets the value of the TEST_SHORT element of this view

 * @return The value which this element has

 */

public short getTEST_SHORT()

{

  return(this.TEST_SHORT);

}

/**

 * Sets the value of the TEST_SHORT element of this view

 * @param value The value to set the element to

 */
```

```
public void setTEST_SHORT(short value)

{

  this.TEST_SHORT = value;

}


/**

 * Gets the value of the TEST_STRING element of this view

 * @return The value which this element has

 */

public String getTEST_STRING()

{

  return(this.TEST_STRING);

}

/**

 * Sets the value of the TEST_STRING element of this view

 * @param value The value to set the element to

 * @throws IllegalArgumentException if the value is too long

 */

public void setTEST_STRING(String value)

{

  if (value.length() > 100)

        throw new IllegalArgumentException("Data too large for

TEST_STRING");

  this.TEST_STRING = value;

}


public void _tmpresend(DataOutputStream encoder)

  throws TPException, IOException {
```

```
    int lcv;
    try {
      encoder.writeInt(TEST_SHORT);
      Utilities.xdr_encode_string_length(encoder,TEST_STRING, 100);
    }
    catch (IOException ie) {
      System.out.println("Error encoding view buffer: " + ie);
    }
    return;
  }



  public void _tmpostrecv(DataInputStream decoder, int recv_size)
    throws TPException, IOException {
    int lcv;
    TEST_SHORT = (short)decoder.readInt();
    TEST_STRING = Utilities.xdr_decode_string(decoder, null);
    return;
  }
}
```

Listing 103 is the corresponding XML document generated using the above VIEW32 class.

Listing 103   Corresponding VIEW332 Class XML Document

```
<ns:VIEW32 xmlns:ns="uri:my-namespace">
```

```
   <View32>

      <TEST_SHORT>1234</TEST_SHORT>

      <TEST_STRING>hello</TEST_STRING>

   </View32>

</ns:VIEW32>
```

Note: The "View32" tag is inserted to represent the sub-type of the view.In this case it is the name of the VIEW32.

Listing 104 shows is the corresponding schema:

Listing 104 Corresponding Schema

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

<xsd:schema

  targetNamespace="uri:my-namespace"

  xmlns="Uri:my-namespace"

  elementFormDefault="unqualified"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="VIEW32">

    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="View32">

          <xsd:complexType>

            <xsd:sequence>

              <xsd:element name="TEST_SHORT" type="xsd:string"/>

              <xsd:element name="TEST_STRING" type="xsd:string"/>

            </xsd:sequence>

          </xsd:complexType>
```

```
        </xsd:element>

      </xsd:sequence>

    </xsd:complexType>

  </xsd:element>

</xsd:schema>
```

Type 3 Conversion

"Type 3" conversion is for STRING, CARRAY, MBSTRING, and X_OCTET Oracle Tuxedo buffer types. You must use wrapped operation. With wrapped operation you must specify the name of the tag to be used to wrap the reply in the `XMLRecord`.

To request `XMLRecord` request buffer, if there is more than one element in the record, a runtime `ResourceException` is thrown. If the request `XMLRecord` is empty or the element contains 0 length string then a 0 length corresponding Oracle Tuxedo buffer type is used to transport the request.

If the reply buffer is CARRAY and you specified `"attachment"`, the CARRAY data is attached to `XMLRecord`. If the reply buffer is CARRAY and you did not specify `"attachment"`, the reply data is treated as an XML document.

Example 1:

If you select `"wrapped"` operation with an XML element tag named `"MYSTRING"`, the reply is wrapped as follows:.

```
<MYSTRING>abc123</MYSTRING>
```

Example 2:

If you do not select the `"wrapped"` operation, the Tuxedo JCA Adapter treats the reply as a native XML document and uses it to create `XMLRecord`. If the data is not a well formed XML document, a `ResourceException` is thrown.

ExampleFML32 Schema With Embedded FML32 and VIEW32

Listing 105 is the XSD file for a FML32 buffer type with embedded FML32 field named `TEST_FML32`, and the embedded View32 field named `TEST_VIEW32`. The `TEST_VIEW32` VIEW32 field has a sub-type `myview32`.

Listing 105   FML32 Schema With Embedded FML32 and VIEW32

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  targetNamespace="uri:my-namespace"
  xmlns="uri:my-namespace"
  elementFormDefault="unqualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="FML32" type="MyBaseFML32Type"/>
 <xsd:complexType name="MyBaseFML32Type">
   <xsd:sequence>
    <xsd:element name="TEST_CHAR" minOccurs="0" type="xsd:byte" />
    <xsd:element name="TEST_STRING" minOccurs="0" type="xsd:string" />
    <xsd:element name="TEST_SHORT" minOccurs="0" type="xsd:short" />
    <xsd:element name="TEST_FML32" minOccurs="0" type="MyEmbeddedFML32Type"
/>
    <xsd:element name="TEST_VIEW32" minOccurs="0"
type="MyEmbeddedView32Type" />
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="MyEmbeddedFML32Type">
   <xsd:sequence>
    <xsd:element name="TEST_STRING" minOccurs="0" type="xsd:string"/>
    <xsd:element name="TEST_SHORT" minOccurs="0" type="xsd:short"/>
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="MyEmbeddedView32Type">
   <xsd:sequence>
```

```
<!--

      "myview32" is the VIEW32 class name, and subtype of VIEW32 buffer

      A VIEW32 def file must be created and use it to create table

      The definition must corresponds to myviewType

      and a viewj32 must be used to generate the class

      the compiled java class must be put in the system classpath

   -->

   <xsd:element name="myview32" minOccurs="0" type="myviewType"/>

  </xsd:sequence>

 </xsd:complexType>

 <xsd:complexType name="myviewType">

  <xsd:sequence>

   <xsd:element name="V_CHAR" minOccurs="0" type="xsd:byte"/>

   <xsd:element name="V_SHORT" minOccurs="0" type="xsd:short"/>

   <xsd:element name="V_STRING" minOccurs="0" type="xsd:string"/>

  </xsd:sequence>

 </xsd:complexType>

</xsd:schema>
```

See Also

- Oracle Tuxedo Adapter Inflow Transaction Guide

- Oracle Tuxedo JCA Adapter Programming Guide

- Oracle Tuxedo JCA Adapter Reference Guide

- Oracle Tuxedo Reference Guide
  DMCONFIG and GWTDOMAIN