ORACLE

Deployment and Configuration

Version 10g Release 3 (10.3)

Deployment and Configuration

This document illustrates how to deploy the Java components generated for a cartridge into Simple Runtime.

Simple Runtime

Simple runtime is more or less a fairly straightforward Java application. It is not a server based application and infact both the client and the runtime along with the generated application run in the same process.

In brief, this is how the Simple runtime works:

1. It is initialized when the client code tries to lookup something for the first time. It prepares a list of deployable entities in the classpath (all Jars). It does this by looking for tplus-jar.xml.

2. It creates a global lookup context (hashtable), where it binds all the all the entities specified in each of the tplus-jar.xml. When the client tries to lookup an entity, it uses this hastable for lookup, but not JNDI.

3. It uses its own connection pooling mechanism to establish DB connection. It uses a file 'data-sources.xml' which contains the properties for the data source. This file should be present in the class path for establishing database connection.

4. It uses the logging configuration defined in the 'log.xml' file for logging messages.

NOTE

As the Simple runtime is not a server based application, the client code should not use server specific features (like assuming JNDI).

As both the client and the runtime run in the same process, the client is not allowed to lookup out process remote objects.

See Also:

Transaction Support in Simple Runtime
Data Source Configuration
Logging Configuration
Simple Runtime with Data Sources from Application Server

Transaction Support in Simple Runtime

Simple runtime supports simple transactions (Non-XA). Thus, a message flow with Transaction requirements can be tested using Simple runtime.

For providing transaction support, the Simple runtime uses a Transaction Manager and it works based on thread local storage (as most TMs do).

To use transaction support in database context, the connection pool class **'com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool'** should be used instead of the class **com.tplus.transform.util.sql.connection.JDBCDriverConnectionPool.** This is required as the resource (in this case, the connection pool implementation) should work with the Transaction Manager. The later class does not support transactions.

These are the limitations of Transaction implementation:

NO XA support
No support for transaction propagation through RMI (only in proc).

See Also:

Simple Runtime
Data Source Configuration
Logging Configuration
Simple Runtime with Data Sources from Application Server

Data Source Configuration

All you need to connect to DB is to create a data-sources.xml file that defines the <data-source> tag with the correct values for the connection attributes and copy it into the generated 'java' directory before running the sample.

The sample given below connects to HSQL database.

```
<?xml version="1.0"?>
<data-sources>
 <data-source
   class="com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool"
   name="invoicedb"
   location="invoicedb"
   connection-driver="org.hsqldb.jdbcDriver"
   url="jdbc:hsqldb:hsql://localhost"
   username="sa"
   password=""
```

```
 />
</data-sources>
```

The following table describes the connection attributes used in the <data-source> tag.

class	Name of the connection pool class. Use the connection pool class **com.volante.component.server.jdbc.ManagedJDBCDriverConnectionPool**, which works with the Transaction Manager to provide transaction support. The class **com.tplus.transform.util.sql.connection.JDBCDriverConnectionPool** can also be used. But this does not support transactions.
name	Name of the data source. Used mainly for error reporting.
location	This should be same as the value of the Data Source property set as part of code generation settings.
connection-driver	JDBC driver class. Please note that this class should be present in the class path for establishing database connection.
url	URL to be passed to the driver
username	User name to be used to get a connection to the database.
password	Password to connect to the database (for the specified username)

See Also:

Simple Runtime
Transaction Support in Simple Runtime
Logging Configuration
Simple Runtime with Data Sources from Application Server

Logging Configuration

To enable logging while running the application, you need to create a log.xml file that defines the <log4j:configuration> tag as shown below and copy it into the generated 'java' directory before running it.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration
xmlns:log4j="http://jakarta.apache.org/log4j/">
     <appender name="consoleLog"
class="org.apache.log4j.ConsoleAppender">
       <param name="Threshold" value="info" />
```

```
          <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern"  value="[%5p] %m%n"/>
          </layout>
     </appender>
     <root>
          <appender-ref ref="ConsoleLog" />
     </root>
</log4j:configuration>
```

Here, it uses the console appender to display log messages in Console.

As the value of the **Threshold** parameter is set to **info**, the application will display information messages while running. The other acceptable values for this parameter are **debug** and **warn**.

See Also:

Simple Runtime
Transaction Support in Simple Runtime
Data Source Configuration
Simple Runtime with Data Sources from Application Server
FAQ

Simple Runtime with Data Sources from Application Server

We can use the data-sources resource defined using Application server instead of providing the full definition in data-sources.xml. We still need to configure a datasource in data-sources.xml. But, Instead of specifying the connection info you just point to server datasource.

Using the server defined resources without using its Transaction Manager does not make sense. We should necessarily lookup and use server's Transaction Manager.

Data Sources Configuration from WebLogic Server

Following data-sources configuration uses data-sources resource defined using WebLogic server.

```
<?xml version="1.0"?>
<data-sources>
    <transaction-manager server-location="javax.transaction.TransactionManager"
/>
     <data-source
class="com.volante.component.server.jdbc.ManagedDataSourceConnectionPool"
        name="hsql"
```

```
        server-location="serverdb"
        location="transformdb"
    />
</data-sources>
```

"**javax.transaction.TransactionManager**" is the Transaction Manager used for WebLogic server.

See Also:

Simple Runtime
Transaction Support in Simple Runtime
Data Source Configuration

FAQ

1. Fundamentally what is the difference between EJB and non-EJB deployment (SimpleRT), within the application server?

   Using SimpleRT inside an EJB server is an interesting possibility. In case of simple rt, there is no concept of server. So the client executes first before the runtime gets a chance. This is how simplert works,

2. When you try to lookup something for the first time (from client code), it prepares a list of deployable entities in the classpath (all Jars). It does this by looking for tplus-jar.xml.

   > It creates a global lookup context (hashtable), where it binds all the all the entities specified in each of the tplus-jar.xml

   Since simplert is more or less fairly straightforward Java, you can in theory use it anywhere you want. But if you are embedding it in an EJB you should be aware of how it works under the covers (since it works based on classpath of its classloader).

   If the generated cartridge is running under SimpleRT, it **does not** have any knowledge of the enclosing EJB server. The implications are,

   > It will not be able to access the application server's DataSources.
   > It will not be able to access the application server's queue or other resources.
   > Does not use the EJB server transactions(this is obvious because it does not even use the resources provided by the Server).

   If simplert is used in a servlet, some of the concerns raised above disappear. You probably want the simplert to be oblivious of the server environment.

3. When simpleRT is depoyed in J2EE server, does it use its own lookup implementation or the server's implementation of JNDI? Can the user choose the implementation?

   SimpleRT uses its own implementation of LookupContext, which is based on a global Hashtable. This also applies to resources. The simple rt lives in its own world and ignores the application completely.

   The user cannot choose the implementation to be used.

4. If one uses simpleRT in library mode (i.e., linked to an application not deployed in J2EE server) how is DB connection established. Does it use its own connection pooling mechanism? In case simpleRT is deployed on J2EE server, can the user choose to use the server's connection pool rather than connecting to DB directly?

   SimpleRT uses its own connection pooling mechanism to establish DB connection. simpleRT looks for a data source specified in the 'Data Source' property set in the code generation settings of the cartridge. It uses a file 'data-sources.xml' which contains the properties for the data source. This file should be present in the class path for establishing database connection.

   If you deploy simpleRT on J2EE server you cannot choose to use the server's connection pool.

See Also:

Deployment and Configuration