BEA WebLogic Integration™

RDBMS Event Generator User Guide

Version 8.5 Service Pack 5
Revised: October 2005

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

Index

CHAPTER 1

Introducing the BEA WebLogic RDBMS Event Generator

This section introduces the BEA WebLogic RDBMS Event Generator.

It includes the following topics:

About the BEA WebLogic RDBMS Event Generator

The RDBMS Event Generator is one of the WebLogic Integration event generators that you can create from the WebLogic Integration Administration Console.

The RDBMS Event Generator uses triggers to detect changes to a database table for added, deleted, or updated rows and publishes the results to Message Broker channels. You can also use the RDBMS Event Generator to run custom queries on the database table and publish the results to Message Broker channels.

To learn more about the event generators available from the WebLogic Integration Administration Console and for information on defining channel rules for the RDBMS Event Generator, see *Managing Weblogic Integration Solutions*.

Additional information regarding the configuration of event generators is also available in the following sections of *Deploying WebLogic Integration Solutions*.

- "Key Deployment Resources" in the Introduction provides information about event generator resources.
- "Deploying Event Generators" in Understanding WebLogic Integration Clusters provides information about deploying event generators in a clustered environment.

Supported RDBMS Events

The RDBMS Event Generator enables integration with RDBMS by retrieving data from a database as XML or a CachedRowSet that can be used in a business process. This provides a convenient and simple method for integrating databases with enterprise applications using WebLogic Integration.

The RDBMS Event Generator supports asynchronous message retrieval from databases, including Oracle, Microsoft SQL Server, IBM DB2 UDB, Informix Dynamic Server, and Sybase Adaptive Server in 2 ways:

- Integration of table event (Insert/Delete/Update) operations in processes
- Integration of custom SQL query output in processes

Supported Databases

The following databases are currently supported:

- DB2
- IBM Informix Dynamic Server
- MS SQL 2000
- Sybase
- Oracle

For updated information and details on supported database versions, see Supported Database Configurations section, in the *Supported Configurations for WebLogic Integration 8.5* document.

Supported Drivers

This section lists the drivers that the BEA WebLogic RDBMS Event Generator supports on the following operating systems:

- Windows 2000 Professional

- Sun Solaris 8 and 9
- HP-UX 11i
- AIX

It also lists the driver class names and JDBC URLs corresponding to the drivers.

Note: The BEA WebLogic RDBMS Event Generator should be used in conjunction with a JDBC 2.1 compliant driver.

Oracle Thin Driver

The Oracle 10g 10.1.0.2.0 Thin Driver file (`WL_HOME/server/lib/ojdbc14.jar`) is shipped with the product.

Table 1-1 Driver Class Names and JDBC URLs for Oracle Thin Driver

For Normal Connection	Driver Class Name	`oracle.jdbc.driver.OracleDriver`
	JDBC URL	`jdbc:oracle:thin:@`*`hostname`*`:`*`port`*`:`*`schema_name`*
	Example	`jdbc:oracle:thin:@127.0.0.1:1521:UAT02`
For XA Connection	Driver Class Name	`oracle.jdbc.xa.client.OracleXADataSource`
	JDBC URL	`jdbc:oracle:thin:@`*`hostname`*`:`*`port`*`:`*`schema_name`*
	Example	`jdbc:oracle:thin:@127.0.0.1:1521:UAT02`

MS SQL Server 2000 JDBC Driver

You can download the MS SQL Server 2000 JDBC Driver files (Version :2.2.0037) `msutil.jar`, `msbase.jar`, and `mssqlserver.jar` from the following URL:

`http://www.microsoft.com/downloads/details.aspx?familyid=9f1874b6-f8e1-4bd6-947c-0fc5bf05bf71&;displaylang=en`

Table 1-2 Driver Class Names and JDBC URLs for MS SQL Server 2000 JDBC Driver

For Normal Connection	Driver Class Name	`com.microsoft.jdbc.sqlserver.SQLServerDriver`
	JDBC URL	`jdbc:microsoft:sqlserver://`*`hostname`*`:`*`port`*`; SelectMethod=Cursor;DatabaseName=?`
	Example	`jdbc:microsoft:sqlserver://127.0.0.1:1433; SelectMethod=Cursor;DatabaseName=AdapterTesting`
For XA Connection	Driver Class Name	`com.microsoft.jdbcx.sqlserver.SQLServerDataSource`
	JDBC URL	`jdbc:microsoft:sqlserver://`*`hostname:port`*`; SelectMethod=?;DatabaseName=?;ServerName=?`
	Example	`jdbc:microsoft:sqlserver://127.0.0.1:1433; SelectMethod=cursor;DatabaseName=AdapterTesting; ServerName=itpl-025019`

DB2 JDBC Driver (JDBC 2.0 Compliant)

The DB2 JDBC Driver file, `db2java.zip`, is available with the database instance.

Table 1-3 Driver Class Names and JDBC URLs for DB2 JDBC Driver (JDBC 2.0 Compliant)

For Normal Connection	Driver Class Name	`COM.ibm.db2.jdbc.app.DB2Driver`
	JDBC URL	`jdbc:db2:`*`DatabaseName`*
	Example	`jdbc:db2:DWCTRLDB`
For XA Connection	Driver Class Name	`COM.ibm.db2.jdbc.DB2XADataSource`
	JDBC URL	`jdbc:db2;DatabaseName=?`
	Example	`jdbc:db2;DatabaseName=DWCTRLDB`

Informix Dynamic Server JDBC Driver

The Informix JDBC Driver Version 2.21 files, `ifxjdbc.jar` and `ifxjdbcx.jar`, are available with the database instance.

Table 1-4 Driver Class Names and JDBC URLs for Informix JDBC Driver

For Normal Connection	Driver Class Name	`com.informix.jdbc.IfxDriver`
	JDBC URL	`jdbc:informix-sqli://`*`host`*`:`*`port`*`/`*`DatabaseName`*`: INFORMIXSERVER=`*`ServerName`*
	Example	`jdbc:informix-sqli://127.0.0.1:1526/BEADEV: INFORMIXSERVER=BEA_SVR_INFX`
For XA Connection	Driver Class Name	`com.informix.jdbcx.IfxXADataSource`
	JDBC URL	`jdbc:informix-sqli://`*`host`*`:`*`port`*`/`*`DatabaseName`*`; ServerName=`*`ServerName`*`;PortNumber=`*`port`*`; IfxIFXHOST=`*`host`*`;DatabaseName=`*`DatabaseName`*
	Example	`jdbc:informix-sqli://127.0.0.1:1526/BEADEV;Serv erName=BEA_SVR_INFX;PortNumber=1526; IfxIFXHOST=127.0.0.1;DatabaseName=BEADEV`

Sybase DataDirect Connect for JDBC

For integration with Sybase Adaptive Server databases, use the BEA WebLogic RDBMS Event Generator with DataDirect Connect for JDBC (Release 3.3). The required JDBC Driver files, `util.jar`, `base.jar`, and `sybase.jar`, are part of DataDirect Connect for JDBC Release 3.3.

For information about obtaining DataDirect Connect for JDBC, see the following URL:

http://www.datadirect.com/products/jdbc/jdbcindex.asp

Table 1-5 Driver Class Names and JDBC URLs for Informix JDBC Driver

For Normal Connection	Driver Class Name	`com.ddtek.jdbc.sybase.SybaseDriver`
	JDBC URL	`jdbc:datadirect:sybase://`*`host`*`:`*`port`*`; databaseName=`*`name`*
	Example	`jdbc:datadirect:sybase://127.0.0.1:2048; databaseName=DB1`

Table 1-5 Driver Class Names and JDBC URLs for Informix JDBC Driver

For XA Connection	Driver Class Name	`com.ddtek.jdbcx.sybase.SybaseDataSource`

	JDBC URL	`jdbc:datadirect:sybase://`*`host`*`:`*`port`*`;`
		`DatabaseName=`*`name`*`;ServerName=`*`host`*`;`
		`PortNumber=`*`port`*`;SelectMethod=cursor`

	Example	`jdbc:datadirect:sybase://127.0.0.1:2048;`
		`DatabaseName=DB1;ServerName=127.0.0.1;`
		`PortNumber=2048;SelectMethod=cursor`

CHAPTER 2

RDBMS Events

This section provides a brief overview of RDBMS events and provides information you should consider before you begin creating data sources and RDBMS events, and defining channel definition rules.

This section covers the following topics:

- RDBMS Events
- Supported Event Types
- General Information
- Architecture Overview

RDBMS Events

Events are asynchronous, one-way messages received from an RDBMS. Events post an XML document/XML as Java String/Serialized CachedRowSet from an RDBMS whenever a specific event of interest is triggered. The event could contain data about a Select, Insert, Update, or Delete operation that has occurred on a table in the RDBMS.

Supported Event Types

The RDBMS Event Generator supports the following event types:

- **Trigger** - A Trigger event provides notification of an Insert, Update, or Delete event occurring in a database table.

- **Query/Post Query** - A Query/Post Query event notifies records of interest based on a *Select Query* given on a database table and executes the SQL specified in the Post Query for each event posted.

General Information

Before you begin creating data sources and RDBMS events, and defining channel definition rules, consider the following:

- You can configure a maximum of 3 Trigger type events per Table - 1 each for Insert, Delete, and Update events.
- Trigger type events have only been tested on Tables. They have not been tested on View, Synonyms, etc.
- Default JMS connection factories must be present for event generators to work.
- RDBMS event generators log errors, warnings, and other messages to a file called `rdbmsEG.log` in the server's domain directory. Refer to this file to troubleshoot RDBMS event generator problems.
- When you are working with Trigger type events, once you have configured a channel rule, a trigger keeps copying the event type (Insert/Delete/Update) rows into another Table from the Table you specified when you defined the channel rule. If you delete the channel rule while it is still publishing events, data loss may occur. To prevent any data loss, you should only delete the channel rule definition when it has stopped publishing events and when there is no event activity on the Table you specified when you defined the channel rule.

When you are configuring a Query/Post Query event type, remember:

- If Post Query is either "no-op" (No Operation) or SQL (which does not delete the Selected Rows), then:
  - The first "x" rows will always keep getting published, where, "x" is Max Rows Per Poll.
  - Sybase ignores the Max Rows Per Poll and returns all the rows matching the SELECT Query in every poll.
  - For DB2 and Informix, automatic-delete is not supported.
  - You must specify the correct DELETE Post Query.
- Post Query is executed for each row returned by SELECT Query.

- Post Query for a row is executed after it gets published to Message Broker.
- If Post Query is vague, for example “DELETE FROM USER_TBL” and SELECT Query is “SELECT FIRST_NAME, LAST_NAME WHERE STATUS=‘TEMP’”, then all rows will be deleted after publishing only the first row.
- The @ prefixed variables in the Post Query works just like “bind variables” in PreparedStatements.

Architecture Overview

The RDBMS Event Generator is designed to replace the RDBMS Adapter's events features. (The RDBMS Adapter Service feature is not a part of the RDBMS Event Generator). This section provides a brief overview of RDBMS Event Generator architecture and presents some of the benefits that the RDBMS Event Generator has over the RDBMS Adapter.

The major difference between the RDBMS Event Generator and the RDBMS Adapter is in the architecture of Trigger Type events. Query/Post Query Type events are essentially the same.

For Trigger Type events, the RDBMS Adapter runs in a single thread and polls and publishes table rows in a single thread. If the publish operation takes longer than the poll interval to complete, then subsequent polls are delayed. Since this is essentially a single-threaded approach, the maximum throughput is limited. In contrast, the RDBMS Event Generator utilizes a multi-threaded approach. Polling and publishing are performed by separate processes. This way, even if some of the publishing threads are delayed, polling can continue and the free threads can publish new rows. Also, you can specify the number of processing/publishing threads to match the rate-of-activity on the database table.

The following example uses an “Insert Trigger Type Event” to illustrate this polling benefit. It takes 10 seconds for 2 threads to process and publish 50 Rows each. This equates to a throughput of (50 x 2)/10 = 10 rows per second, or 10 x 60 x 60 = 36,000 rows per hour. To make maximum use of the processing threads occupied all the time and to always obtain a consistent throughput, a polling interval of 7-8 seconds is selected. The database table on which this Trigger Event has been configured must have at least 100 rows being Inserted every 7-8 seconds to obtain this constant throughput of 10 rows per second.

You should only use the above example as a thumb-rule to help configure events. The actual numbers depend on several factors like application server hardware, database hardware, data types chosen for publication, number of columns chosen for publication, application server execute queue size, etc.

Clusters

In a cluster, polling is performed by each managed node in a round-robin fashion. Processing/Publishing of the polled rows is also load-balanced in a round-robin fashion. The Connection Pool and the Data Source must be targeted to the entire Cluster. For more information on deploying RDBMS Event Generators in clusters, see Event Generator Resources in *Deploying WebLogic Integration Solutions*.

CHAPTER 3

Creating a Data Source

Before you can create RDBMS events and define channel rules for your new events, you must create a data source that points to the database and hence the Table on which the channel rule (Event) will be defined. For more information on defining channel rules for RDBMS Event Generators, see "Defining Channel Rules for a RDBMS Event Generator" in *Managing WebLogic Integration Solutions Guide*.

After you create your data source, you must also create the message broker channels that the RDBMS event publishes to, before you can define channel rules for any RDBMS events.

The following sections detail how you create a data source and message broker channels:

- Creating a Data Source
- Creating Message Broker Channels

Creating a Data Source

Before you create a data source, you must be aware of the database connection parameters for the data source you want to connect to. If you are unaware of the database connection parameters, contact your system administrator.

The steps involved in creating a data source are:

- Adding the Driver to the Classpath
- Creating your Data Source Connection
- Defining the Connection Pool Configuration

- Configuring a New Data Source

Adding the Driver to the Classpath

The first step in creating a data source is adding the required driver to the WebLogic classpath.

To add a driver to the WebLogic classpath:

1. Locate the `<bea_home>\weblogic81\samples\domains\<domain name>` directory.
2. Open the `setDomainEnv.cmd` and scroll to the bottom of the file where is reads:

   ```
   @REM SET THE CLASSPATH
   set CLASSPATH=%PRE_CLASSPATH%;%WEBLOGIC_CLASSPATH...
   ```

3. Add the following statement immediately AFTER `@REM SET THE CLASSPATH`:

   ```
   set CLASSPATH=%WL_HOME%\server\lib\driver.jar;%CLASSPATH%
   ```

   where `driver.jar` is the jar file for the database driver that you want to add to the classpath. For example, for an Oracle driver, the command would read:

   ```
   set CLASSPATH=%WL_HOME%\server\lib\ojdbc14.jar;%CLASSPATH%
   ```

4. Save the file.

Note: Before you use a MS SQL XA driver to create events, you must follow the steps detailed in "Installing Stored Procedures for JTA" in *The MS SQL Service Driver*.

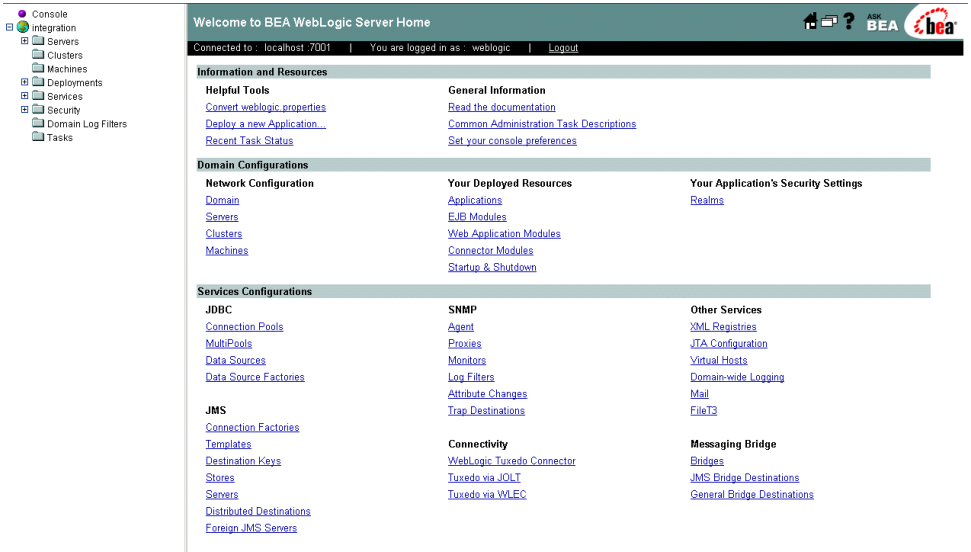
Creating your Data Source Connection

The next step in creating a data source is creating the connection that you will use to connect to the data source.

To create a data source connection:

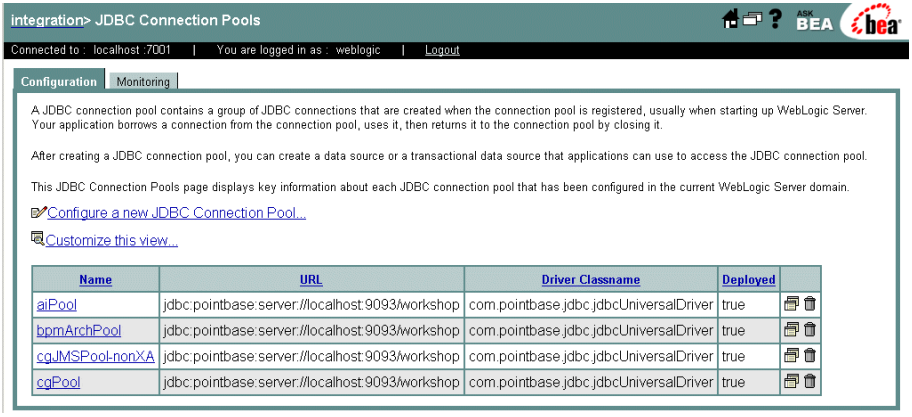
1. Start the WebLogic Server using the `startWebLogic.cmd` command or from WebLogic Workshop.
2. Launch the WebLogic Server Console by entering http://localhost:7001/console in a Web browser or by selecting **Tools→WebLogic Server→WebLogic Console...**.
3. Enter your username and password and click **Sign In**. The default username and password are both "weblogic."

   The WebLogic Server **Home** page is displayed, as shown in the following figure.



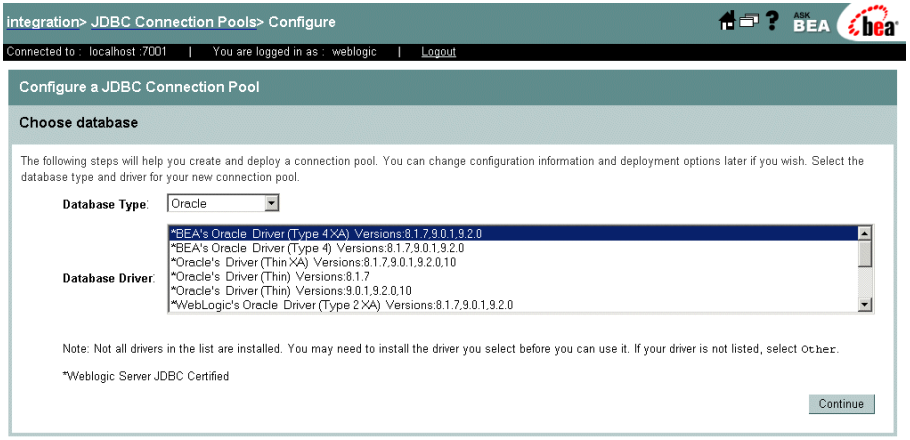
4. Click **Connection Pools** under **JDBC** in the **Service Configurations** section.

   The **JDBC Connection Pools** page is displayed, as shown in the following figure.



5. Click **Configure a new JDBC Connection Pool**.

   The **Configure JDBC Connection Pool** page is displayed, as shown in the following figure.



6. Select the database type that you are going to connect to from the **Database Type** drop-down list.

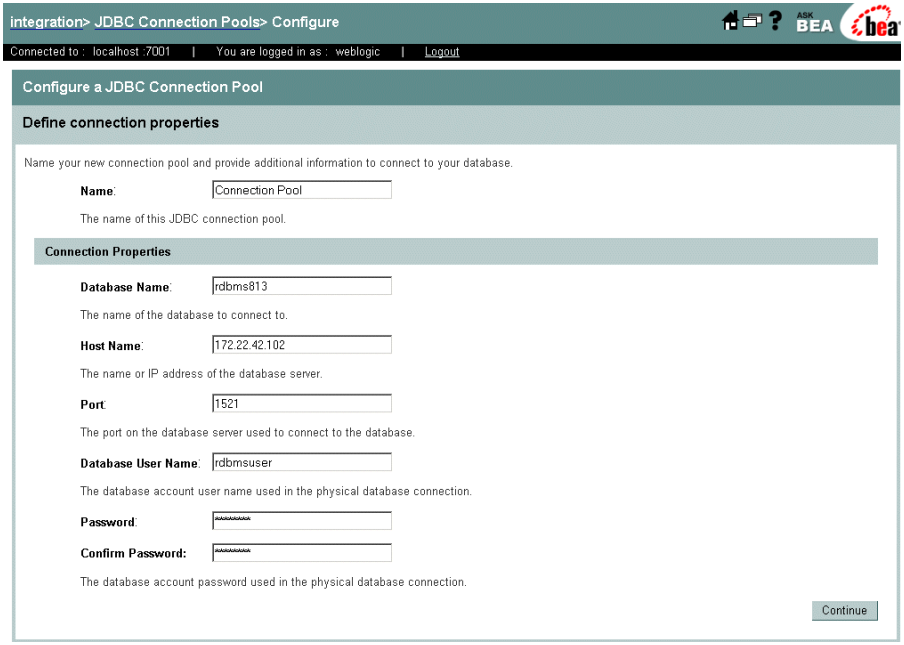
   For a list of the supported databases, see "Supported Databases" on page 1-2.

7. Select the database driver you want to use from the **Database Driver** drop-down list.

   Note: WebLogic Server JDBC certified drivers are marked with *.

8. Click **Continue**.

   The **Define Connection Properties** page is displayed, as shown in the following figure.

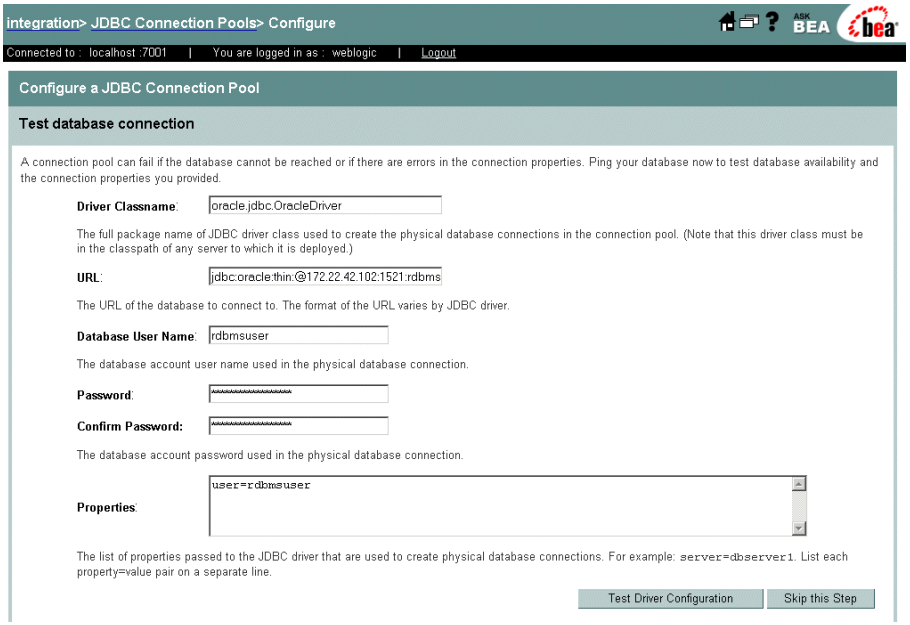


The fields that appear on this page depend on the database and driver selected on the previous page.

9. Enter a name for your connection in the **Name** field.
10. Enter the other connection details and click **Continue**.

    Note: If you want to configure a trigger type event, it is recommended that the database account user name you enter in the **Database User Name** field is the same as the schema name of the table on which you want to configure the trigger type event. The account name you use must also have permission to CREATE/DROP tables and triggers. If you are configuring an Oracle database, the account name you use must also have permission to CREATE/DROP tables, triggers, and sequences.

    The **Test Database Connection** page is displayed, as shown in the following figure.



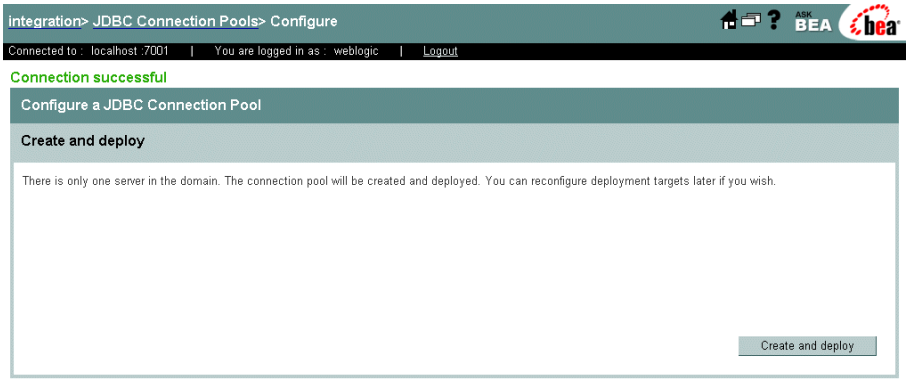
The fields that appear on this page depend on the database and driver selected on the previous page.

11. Make sure the details displayed on this page are correct and click **Test Driver Configuration**.

    It is recommended that you test the driver configuration before proceeding to the next step. If you do not test the driver configuration and you have made an error in the previous steps, you will not be able to connect to the required database when you try to configure your RDBMS Event Generator.

    An error message will appear at the top of the page if there is a problem with your connection settings.

    When your configuration passes the test, the **Create and Deploy** page is displayed, as shown in the following figure.



12. If there is more than one server in the domain you specified, check the server you want to deploy on and click **Create and deploy**.

    The connection is created and deployed and you are returned to the **JDBC Connection Pools** page.

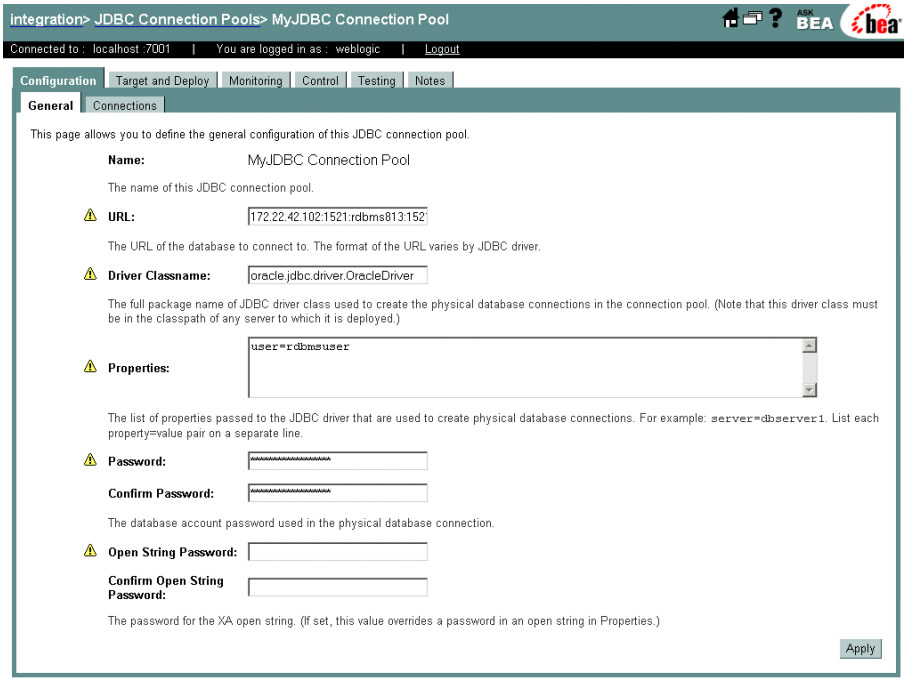
Defining the Connection Pool Configuration

The next step is defining the connection pool configuration for your data source.

To define the connection pool configuration:

1. On the **JDBC Connection Pools** page click the name of your new connection.

   The **Connection Pool Configuration** page is displayed, as shown in the following figure.



2. Click the **Connections** tab.
3. Enter your configuration details in the fields provided.
4. If you are using a XA driver, click **Show** in the **Advanced Options** section.
5. Make sure that the **Supports Local Transaction** option is selected.
6. Click **Apply**.

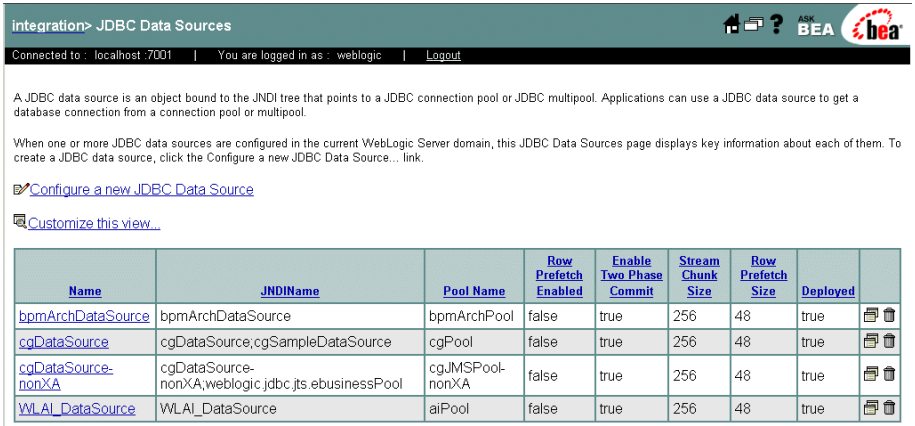
Configuring a New Data Source

The final step in creating your data source connection is creating a new data source.

To create a new Data Source:

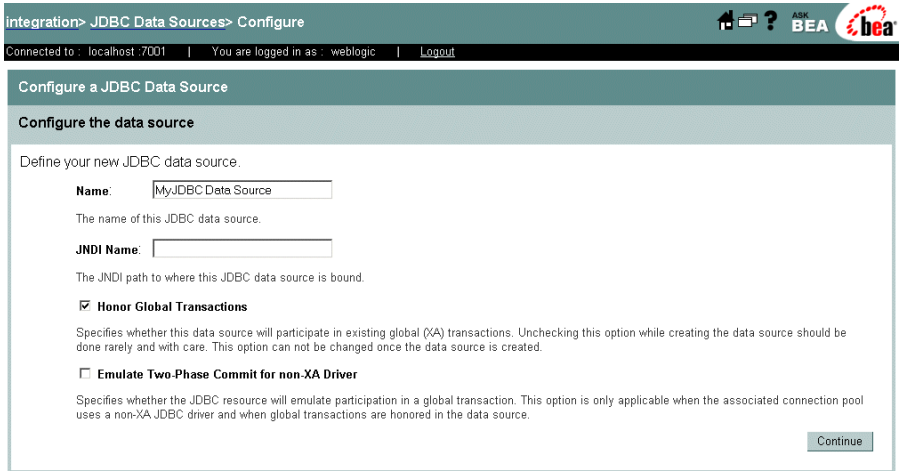
1. On the WebLogic Server **Home** page, click **Data Sources** under **JDBC** in the **Service Configurations** section.

   The **JDBC Data Sources** page is displayed, as shown in the following figure.



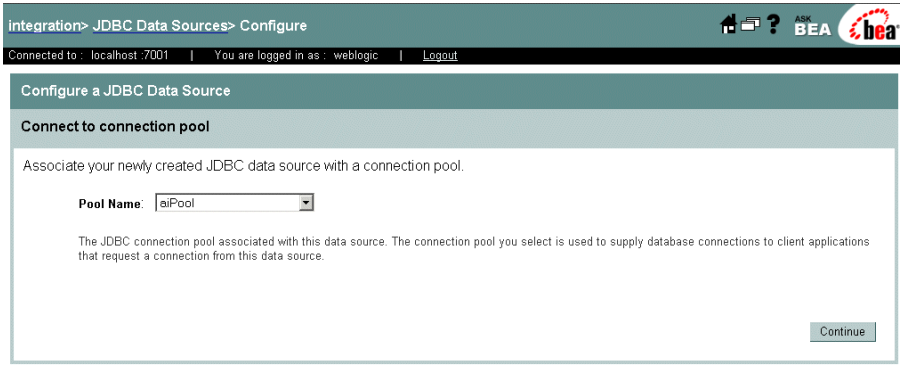
2. Click **Configure a new JDBC Data Source**.

   The **Configure the Data Source** page is displayed, as shown in the following figure.



3. Enter a name for the new data source in the **Name** field.
4. In the **JNDI Name** field, enter the JNDI path to where the data source is bound.
5. If you are using a non-XA driver, check the **Emulate Two-Phase Commit for non-XA Driver** checkbox.
6. Click **Continue**.

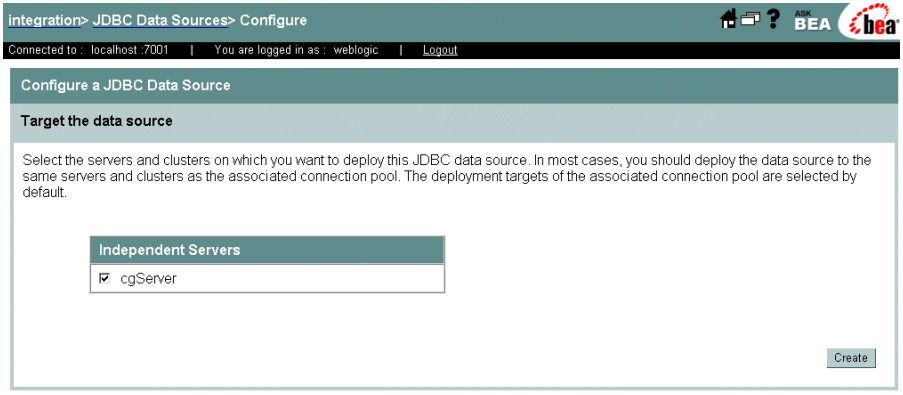
   The **Connect to Connection Pool** page is displayed, as shown in the following figure.



7. Select the connection you created in "Creating your Data Source Connection" on page 3-2 from the **Pool Name** drop-down list.

8. Click **Continue**.

   The **Target the Data Source** page is displayed, as shown in the following figure.



9. Select the servers and clusters on which you want to deploy this JDBC data source. In most cases, you should deploy the data source to the same servers and clusters as the associated connection pool. The deployment targets of the associated connection pool are selected by default.

10. Click **Create**.

    The new data source is created and you are returned to the **JDBC Data Sources** page.

Before you can create your RDBMS Event Generators in the WebLogic Integration Administration Console and define channel rules for these new event generators, you must create the message broker channels to which your new event generators will publish.

Notes on Creating a Data Source

You should consider the following:

- For DB2, Oracle, and MS SQL databases, the event generator uses `DetabaseMetadata.getUserName()` as the schema name to create database artifacts for Tables, Triggers, and Sequences (Oracle only).
- If you configure two tables with the same name in different schemas, you will get a `StringIndexOutOfBoundsException`. For example, the exception will occur if there are 2 tables with the same name in different schemas, like `MASAA01.prod` and `dbo.prod` in the same Catalog with the login name MASAA01, and the user creates an event on `dbo.prod`.

  The exception will occur in this case, since a MS SQL Stored procedure called 'sp_columns' is used to retrieve the columns in the Table and their data types. The stored procedure takes only the Table name and not the fully qualified Table name. So it always returns the columns from `MASAA01.prod` instead of the `dbo.prod` columns, even though the `dbo.prod` columns should be received.
- If you want to create Trigger type events on a Sybase database, you must configure the events on the `dbo` schema.

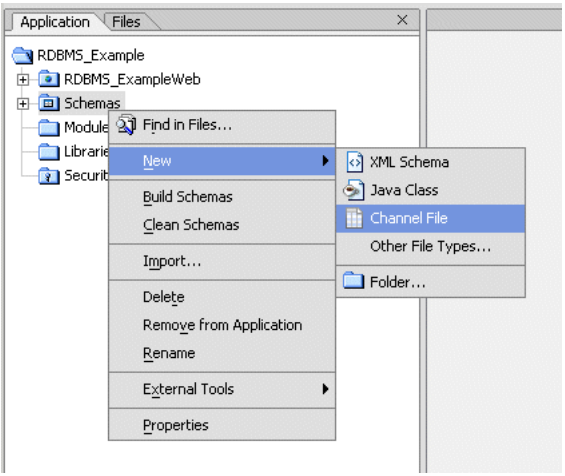
Creating Message Broker Channels

This section describes how you create the message broker channels that your RDBMS Event Generator will publish to. You need to create three different types of message broker channel:

- XML channel
- String channel
- RawData channel

To create message broker channels:

1. Start WebLogic Workshop and ensure that the server is running.
2. Click **File→New→Application** and provide a name for the application.
3. In the **Application** pane, right-click the **Schemas** folder and select **New→Channel File**.



4. Enter a name for the channel and click **Create**.

   The code for the new channel is displayed.

5. If you have a pre-configured XML channel file, replace the code for the newly created channel with your own custom code. If you do not have your own custom code, you can use the code displayed below:

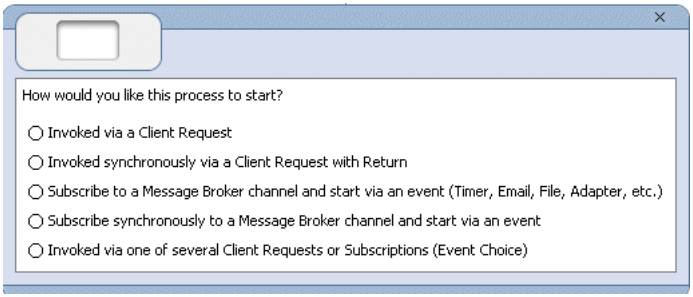
```
<?xml version="1.0"?>

<!--

        A sample channel file

        Change "channelPrefix" and <channel/> elements to customize

        The following namespaces are used in this sample file. These

        namespaces refer to schemas that must be present in a schema

        directory.

-       xmlns:eg=http://www.bea.com/wli/eventGenerator" is used

        for event generator metadata references

-       xmlns:dp="http://www.bea.com/wli/control/dynamicProperties" is

        used by the file event generator to pass payload for pass-by-filename

-       xmlns:oagpo="http://www.openapplications.org/003_process_po_007" is used
for a sample payload description

-->
```

```
<channels xmlns="http://www.bea.com/wli/broker/channelfile"
                channelPrefix="/SamplePrefix"
                xmlns:eg="http://www.bea.com/wli/eventGenerator"
               xmlns:dp="http://www.bea.com/wli/control/dynamicProperties"

xmlns:oagpo="http://www.openapplications.org/003_process_po_007">
        <channel name ="Samples" messageType="none">

                <!-- A simple channel passing XML -->

                <channel name ="SampleXmlChannel" messageType="xml"/>


                <!-- A simple channel passing rawData -->

               <channel name ="SampleRawDataChannel" messageType="rawData"/>


                <!-- A simple channel passing a string -->

                <channel name ="SampleStringChannel" messageType="string"/>

        </channel>
</channels>
```

6. Click **File→New→Process File**.
7. Name the new file and click **Create**.
8. Double-click this new process file.

9. Double-click the Starting Event node.

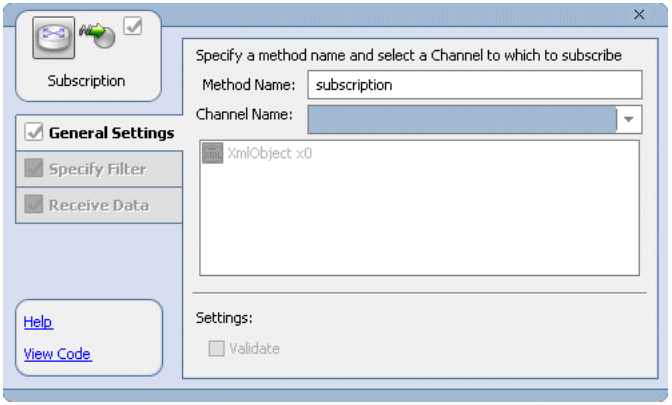
   The **Node Definition** dialog is displayed.



10. Select the **Subscribe to a Message Broker Channel and start via an event** and close the dialog.

11. Double-click on the node again.

    The **Node Details** dialog is displayed.



12. Select the **SampleXmlChannel** from the **Channel Name** drop-down list. If you have configured your own XMLChannel message broker, select it instead.

13. If desired, you can click **View Code** in the bottom left-hand corner and enter the following `System.out` message:

    ```
    System.out.printIn(System.currentTimeMillis()+"-"+x0);
    ```

14. Click the **Receive Data** tab and enter the variable details.

15. Close the file and click **Build→Build Application**.

This creates the XML channel and the JPD is automatically deployed on the server.

16. To create the string channel, repeat the above steps but select **SampleStringChannel** instead of the **SampleXmlChannel** when you select the **Channel Name**.

17. To create the RawData channel, repeat the above steps but select **SampleRawDataChannel** instead of the **SampleXmlChannel** when you select the **Channel Name**.

For the XML Channel, there is a type-safe way of retrieving the XML contents:

1. Right-click on the Schemas folder in the Workshop project and click **Import**. Select the generated `.XSD` file from the Server domain directory. The `.XSD` is created in a folder with the same name as the channel rule definition.

2. Workshop automatically compiles the `.XSD` file into Java classes

3. For an event having the name `testFirst`, which publishes 2 columns of type `CHAR` and `DATE`, `TableRowSet.xsd` is the generated schema file. The contents of the `subscription(..)` method file outlined below illustrate how to use XmlObject using the generated classes.

```
import com.bea.wli.rdbmsEG.testFirst.TableRowSetDocument;

import com.bea.wli.rdbmsEG.testFirst.TableRowSetDocument.TableRowSet;

import
com.bea.wli.rdbmsEG.testFirst.TableRowSetDocument.TableRowSet.TableRow;

....


public void subscription(com.bea.xml.XmlObject x0)

    {

        //#START: CODE GENERATED - PROTECTED SECTION - you can safely add
code above this comment in this method. #//

        // input transform

        // parameter assignment

        //#END  : CODE GENERATED - PROTECTED SECTION - you can safely add
code below this comment in this method. #//


       final TableRowSetDocument doc = TableRowSetDocument.Factory.newInstance();
```

```
        doc.set(x0);

        System.out.println("The document: " + doc);

        final TableRowSet rowSet = doc.getTableRowSet();
        final TableRow[] rows = rowSet.getTableRowArray();
        for(int i = 0; i < rows.length; i++)
        {
            System.out.println("---" + (i + 1) + "---");
            System.out.println("CHAR1: " + rows[i].getCHAR1());
            System.out.println("DATE1: " + rows[i].getDATE1());
        }
    }
```

For a RawData Channel, the code snippet below illustrates how it can be used to retrieve data published by the RDBMS Event Generator:

```
import com.bea.data.RawData;
import java.io.ByteArrayInputStream;
import java.io.ObjectInputStream;
import weblogic.jdbc.rowset.WLCachedRowSet;

….

    public void subscription(com.bea.data.RawData x0)
    {
        //#START: CODE GENERATED - PROTECTED SECTION - you can safely add
code above this comment in this method. #//
        // input transform
```

```
        // parameter assignment
        //#END  : CODE GENERATED - PROTECTED SECTION - you can safely add
code below this comment in this method. #//

        try
        {
            ByteArrayInputStream arrayInputStream = new
ByteArrayInputStream(rawData.byteValue());
            ObjectInputStream objectInputStream = new
ObjectInputStream(arrayInputStream);
            WLCachedRowSet rowSet = (WLCachedRowSet)
objectInputStream.readObject();

            System.out.println("-- Event --\r\n");
            while (rowSet.next())
            {
                Map map = rowSet.getCurrentRow();

                for (Iterator iterator = map.keySet().iterator();
iterator.hasNext();)
                {
                    Object key = iterator.next();
                    System.out.println("  Column: " + key + ", Value: " +
map.get(key));
                }
            }
        }
        catch(Exception e)
        {
```

```
            e.printStackTrace();
        }
    }
```

For more information on message broker channels, see How do I: Create Message Broker Channels? in *Publishing and Subscribing to Channels* in the WebLogic Workshop Help.

The Message Broker module in the WebLogic Integration Administration Console allows you to monitor and manage all the Message Broker channels in your application. To learn more, see Message Broker in *Managing WebLogic Integration Solutions*.

Once you have created your data source and message broker channels, you can create RDBMS Event Generators and define channel rules for these generators in the WebLogic Integration Administration Console.

For information on creating RDBMS Event Generators and defining channel rules for these generators, see Event Generators in *Managing WebLogic Integration Solutions*.

Supported Data Types

This section lists the supported data types for the RDBMS Event Generator for the Trigger Event Scheme and the Intrusive Event Query Scheme. The non-supported data types are also listed.

The following topics are covered in this section:

Supported Data Types - Trigger Event Scheme

The following table summarizes the supported data types for the RDBMS Event Generator (Trigger Event Scheme).

Table A-1 Supported Data Types (Trigger Event Scheme)

Oracle	Microsoft SQL Server	DB2 UDB	Informix Dynamic Server	Sybase Adaptive Server
• BLOB • CHAR • CLOB • DATE • DECIMAL • FLOAT • INTEGER • NUMBER • NUMERIC • RAW • REAL • SMALLINT • VARCHAR2 • NCLOB • NVARCHAR2 • NCHAR	• BIGINT • BIT • CHAT • DATETIME • DECIMAL • FLOAT • INT • MONEY • NCHAR • NUMERIC • NVARCHAR • REAL • SMALLDATETIME • SMALLINT • SMALLMONEY • TINYINT • VARCHAR	• BIGINT • BLOB • CHARACTER • CLOB • DATE • DECIMAL • DOUBLE • INTEGER • LONGVARCHAR • REAL • SMALLINT • TIME • TIMESTAMP • VARCHAR	• BLOB • BOOLEAN • BYTE • CHAR • CLOB • DATE • DATETIME • DECIMAL • FLOAT • INT • INT8 • INTEGER • MONEY • NCHAR • NVARCHAR • SMALLFLOAT • SMALLINT • TEXT • VARCHAR	• BINARY • BIT • CHAR • DATETIME • DECIMAL • DOUBLE • FLOAT • INTEGER • NCHAR • NUMERIC • NVARCHAR • REAL • SMALLDATETIME • SMALLINT • TINYINT • VARBINARY • VARCHAR • MONEY • SMALLMONEY

Note: In Oracle databases, the driver returns `java.sql.Types.NUMERIC` as the JDBC Type for the following Oracle data types:

- `DECIMAL`
- `FLOAT`
- `INTEGER`
- `NUMBER`
- `NUMERIC`
- `REAL`

The generated XSD maps all these data types to `xsd:integer`.

Note: For the MS SQL driver:

- `NUMERIC(p,s)` returns a JDBC Type of `NUMERIC` which maps to `xsd:integer`

- TIMESTAMP returns a JDBC Type of BINARY which maps to xsd:base64Binary
- UNIQUEIDENTIFIER returns a JDBC Type of CHAR which maps to xsd:string

Supported Data Types - Intrusive Query Event Scheme

In addition to the data types supported by the Trigger Event scheme listed above, the Intrusive Query Event scheme supports the data types listed in the following table:

Table A-2 Supported Data Types (Intrusive Query Event Scheme)

Oracle	Microsoft SQL Server	DB2 UDB	Informix Dynamic Server	Sybase Adaptive Server
• LONG	• BINARY • TIMESTAMP • UNIQUEIDENTIFIER • VARBINARY			

Non-Supported Data Types for RDBMS

Table A-3 Non-Supported Data Types (Trigger Event Scheme)

Oracle	Microsoft SQL Server	DB2 UDB	Informix Dynamic Server	Sybase Adaptive Server
• REFCURSOR • LONGRAW • INTERVAL DAY (day_precision) TO SECOND • INTERVAL YEAR (year_precision) TO MONTH • TIMESTAMP • ROWID • UROWID • BFILE	• NTEXT • IMAGE • TEXT • TIMESTAMP • UNIQUEIDENTIFIER • VARBINARY • BINARY • SQL_VARIANT		• SERIAL • SERIAL8 • INTERVAL • LVARCHAR	• TEXT • IMAGE • TIME • TIMESTAMP • DATE • SYSNAME • UNICHAR • UNIVARCHAR

Index