Data access -- table of contents

Development

Administration

4.2.4.2: Obtaining and using database connections

IBM WebSphere Application Server Version 4.0 provides two options for accessing database connections:

- Programming directly to the connection pooling model through the JDBC 2.0 Optional Package API
- Use of the IBM data access beans, which also use connection pooling but give you additional ability to manipulate result sets

WebSphere Application Server versions earlier than 4.0 also supported the connection manager model, which was based on JDBC 1.0. If your Web applications used the connection manager model, you must migrate these in Version 4.0 to use connection pooling.

IBM WebSphere Application Server also provides data access beans, which offer a rich set of features for working with relational database queries and result sets.

For a comprehensive treatment of WebSphere connection pooling and data access,be sure to read the IBM whitepaper to be published on the Webduring the summer of 2001.

Tips for data access programming

If your product installation usesOracle or Sybase for the repository and you try to use DB2 datasourcesfor enterprise beans,the DB2 JDBC provider might fail to load.To fix the problem, your administrator mustrun the db2profile and usejdbc2 scripts.For details, see the Related information.

For Merant users only: If your bean-managed persistence entity beans are used with`SELECT ... FOR UPDATE` andan isolation level of READ_COMMITTED,you might lose updates when multiple transactionsto a single bean collide.You can avoid collision by using anover-qualified update approach, as follows:

```
UPDATE Employee SET c1 = new_c1_value, SET c2 = new_c2_value   WHERE  (primaryKey = key_value AND c1
= old_c1_value                                  AND c2 = old_c2_value)
```

In this example, old_c1_value and old_c2_valueare the values that were read from the database.new_c1_value and new_c2_valueare the values to be stored back.

Considerations for DB2/390

Speak with your DB2/390 administrator about setting the RRULOCKparameter to `YES`.This ensures that `SELECT ... FOR UPDATE` statements get an update lock rather than a sharable lock.If your database is using sharable locksand you attempt to commit updates later,the database can become deadlocked.

4.2.4.2.1: Accessing data with the JDBC 2.0 Optional Package APIs

In JDBC 1.0 and the JDBC 2.0 Core API, the DriverManager class is used exclusively for obtaining a connection to a database. The database URL, user ID, and password are used in the getConnection() call. In the JDBC 2.0 Optional Package API, the DataSource object provides a means for obtaining connections to a database. The benefit of using datasourcesis that the creation and management of the connection factory is centralized. Applications do not need to have specific information like the database name, user ID, or password in order to obtain a connection to the database.

The steps for obtaining and using a connection with the JDBC 2.0 Optional Package API differ slightly from those in the JDBC 2.0 Core API example. Using the extensions, you access a relational database as follows:

1. Retrieve a DataSource object from the JNDI naming service
2. Obtain a Connection object from the datasource
3. Send SQL queries or updates to the database management system
4. Process the results

The connection obtained from the datasource is a pooled connection. This means that the Connection object is obtained from a pool of connections managed by IBM WebSphere Application Server.The following code fragment shows how to obtain and use a connection with the JDBC 2.0 Optional Package API:

```
try {// Retrieve a DataSource through the JNDI Naming Service       java.util.Properties parms = new
java.util.Properties();   parms.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");      // Create the Initial Naming Context
javax.naming.Context ctx = new javax.naming.InitialContext(parms);      // Lookup through the naming
service to retrieve a DataSource object   javax.sql.DataSource ds =
(javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/SampleDB");     // Obtain a Connection from the
DataSource   java.sql.Connection conn = ds.getConnection(); // query the database
java.sql.Statement stmt = conn.createStatement();   java.sql.ResultSet rs =
stmt.executeQuery("SELECT EMPNO, FIRSTNME, LASTNAME FROM SAMPLE");  // process the results   while
(rs.next()) {      String empno = rs.getString("EMPNO");      String firstnme =
rs.getString("FIRSTNME");      String lastname = rs.getString("LASTNAME");   // work with results
}} catch (java.sql.SQLException sqle) {// handle SQLException} finally {   try {      if (rs !=
null) rs.close();   }   catch (java.sql.SQLException sqle) {   // can ignore   }   try {      if
(stmt != null) stmt.close();   }    catch (java.sql.SQLException sqle) {   // can ignore   }   try {
if (conn != null) conn.close();   }    catch (SQLException sqle) {   // can ignore   }} // end
finally
```

In the previous example, the first action is to retrieve a DataSource object from the JNDI namespace. This is done by creating a Properties object of parameters used to set up an InitialContext object. After a context is obtained, a lookup on the context is performed to find the specific datasource necessary, in this case, SampleDB.

(In this example, it is assumed the datasource has already been created and bound into JNDI by the WebSphere administrator. For information about doing thisin application code, see the Related information.)

After a DataSource object is obtained, the application code calls getConnection()on the datasource to get a Connection object. After the connection is acquired, the querying and processing steps are the same as for theJDBC 2.0 Core API example.

4.2.4.2.1.1: Creating datasources with the WebSphere connection pooling API

IBM WebSphere Application Serverprovides a public API to enable you to configure a WebSphere datasource in application code.This is necessary only when the application must create a datasource on demand.Otherwise, the datasource is configured by the administrator in the administrative console.

The complete API specification can be found in javadoc for the class com.ibm.websphere.advanced.cm.factory.DataSourceFactory. See the Related information.

To create a datasource on demand in an application, the application must do the following:

1. Create a Properties object with datasource properties
2. Obtain a datasource from the factory
3. Bind the datasource into JNDI

The following code fragment shows how an application would create a datasource and bind it into JNDI:

```
import com.ibm.websphere.advanced.cm.factory.DataSourceFactory;...try {   // Create a properties
file for the DataSource   java.util.Properties prop = new java.util.Properties();
prop.put(DataSourceFactory.NAME, "SampleDB");   prop.put(DataSourceFactory.DATASOURCE_CLASS_NAME,
"COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource");   prop.put(DataSourceFactory.DESCRIPTION, "My
sample                datasource");   prop.put("databaseName", "sample");// Obtain a DataSource from
the factory   DataSource ds = DataSourceFactory.getDataSource(prop);// Bind the DataSource into JNDI
DataSourceFactory.bindDataSource(ds);} catch (ClassNotFoundException cnfe) {// check the class path
for all necessary classes} catch (CMFactoryException cmfe) {// Example of exception: incorrect
properties} catch (NamingException ne) {// Example of exception:  datasource by this name may
already exist}
```

To create a datasource for binding into JNDI, the application must firstcreate a Properties object to hold the DataSource configuration properties.The only properties required for the datasource from a WebSphere perspective are:

- NAME -The name of the datasource. This is used to identify the datasource when it is bound into JNDI.
- DATASOURCE_CLASS_NAME - The complete name of the DataSource class that can be found in the JDBC resource archive file(often referred to as the JDBC provider package).This DataSource class will be used to create connections to the database. The class specified here must implement javax.sql.ConnectionPoolDataSource or javax.sql.XADataSource.

However, depending on the DataSource class specified in the DATASOURCE_CLASS_NAME property, there may be other vendor-specific properties required. In this example, the databaseName property is also required,because DB2ConnectionPoolDataSource is being used. For more information on these vendor-specific properties, see the vendor's documentation for the complete list of properties supported for a datasource.

After a properties object is created, the application can create a new DataSource object by calling getDataSource() on the factory, passing in the Properties object as a parameter. This creates an object of type DataSource, but it is not yet bound into JNDI. To bind a datasource into JNDI,call bindDataSource() on the factory.At this point, other applications can share the datasource by retrieving it from JNDI with the name property specified on creation.

All other APIs specific to connection pooling are not public APIs. Applications that use a WebSphere datasource should followthe JDBC 2.0 Core and JDBC 2.0 Optional Package APIs.

4.2.4.2.1.2: Tips for using connection pooling

Most best practices have been documented elsewhere in Related information.The following are additional items that have not been explicitly called out:

Obtain and close connection in the same method. An application should obtain and close its connection in the method that requires the connection. This keeps the application from holding resources not being used and leaves more available connections in the pool for other applications. In addition, this practice removes the temptation to use the same connection in multiple transactions, which, by default, is not allowed. This practice does not cost the application much in performance,because the Connection object is from a pool of connections, where the overhead for establishing the connection has already been incurred.Lastly, make sure to declare the Connection object in the same method as the getConnection() call in a servlet;otherwise, the Connection object works as if it is a static variable(see "Worst Practices" later in this article for problems with this).

If you opened it, close it. All JDBC resources that have been obtained by an application should be explicitly closed by that application. The product tries to clean up JDBC resources on a connection after the connection has been closed. However, this behavior should not be relied upon, especially if the application might be migrated to another platform in the future.

For servlets, obtain DataSource in the init() method. For performance reasons, it is usually a good idea to put the JNDI lookup for the datasource into the init() method of the servlet. Because the datasource is simply a factory for connections that does not typically change,retrieving it in this method ensures that the lookup happens only once.

Worst practices

The following are some very common problems with applications that should be avoided, because they most often result in unexpected failures:

Do not close connections in a finalize() method. If an application waits to close a connection or other JDBC resource until the finalize() method, the connection is not closed until the object that obtained it is garbage-collected. This leads to problems if the application is not very thorough about closing its JDBC resources, such as ResultSet objects. Databases can quickly run out of the memory required to store the information about all of the JDBC resources it currently has open. In addition, the pool can quickly run out of connections to service other requests.

Do not declare connections as static objects. It is never recommended that connections be declared as static objects. If a connection is declared as static, the same connection might get used on different threads at the same time. This causes a great deal of difficulty, within both the product and the database.

In servlets, do not declare Connection objects as instance variables. In a servlet, all variables declared as instance variables act as if they are class variables. For example, in a servlet with an instance variable

```
Connection conn = null;
```

this variable acts as if it were static. In this case, all instances of the servlet use the same Connection object. This is because a single servlet instance can be used to serve multiple Web requests in different threads.

In CMP beans, do not manage data access. If a Container Managed Persistence (CMP) bean is writtenso that it manages its own data access, this data access may be part of a global transaction. Generally, if specialized data access is required,use a BMP session bean.

4.2.4.2.1.3: Handling data access exceptions

For data access, the standard Java exception class to catch is java.sql.SQLException.IBM WebSphere Application Servermonitors for specific SQL exceptions thrown from the database. Several of these exceptions are mapped to WebSphere-specific exceptions. The product provides WebSphere-specific exceptions to ease development by not requiring you to know all of the database-specific SQL exceptions that could be thrown in typical situations. In addition, monitoring SQL exceptions enables the product and application to recover from common problems like intermittent network or database outages.

ConnectionWaitTimeoutException

This exception (com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException) indicates that the application has waited for the connectionTimeout (CONN_TIMEOUT) number of seconds and has not been returned a connection. This can occur when the pool is at its maximum size and all of the connections are in use by other applications for the duration of the wait. In addition, there are no connections currently in use that the application can share, because either the user ID and password are differentor it is in a different transaction.The following code fragment shows how to use this exception:

```
java.sql.Connection conn = null;javax.sql.DataSource ds = null;...try {// Retrieve a DataSource
through the JNDI Naming Service       java.util.Properties parms = new java.util.Properties();
setProperty.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");      // Create the Initial Naming Context
javax.naming.Context ctx = new       javax.naming.InitialContext(parms);      // Lookup through the
naming service to retrieve a DataSource object   javax.sql.DataSource ds =
(javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/SampleDB");   conn = ds.getConnection();
// work on connection} catch (com.ibm.ejs.cm.pool.ConnectionWaitTimeoutException cw) {// notify the
user that the system could not provide a // connection to the database}  catch
(java.sql.SQLException sqle) {// deal with exception}
```

In all cases in which the ConnectionWaitTimeoutException is caught, there is very little to do in terms of recovery. It usually doesn't make sense to retry the getConnection() method, because if a longer wait time is required, the connectionTimeout datasource property should be set higher. Therefore, if this exception is caught by the application, the administrator should review the expected usage of the application and tune the connection pool and the database accordingly.

StaleConnectionException

This exception (com.ibm.websphere.ce.cm.StaleConnectionException) indicates that the connection currently being held is no longer valid. This can occur for numerous reasons, including:

- The application fails to get a connectionbecause of a problem such as the database not being started.
- A connection is no longer usable because of a database failure. When an application tries to use a connection it previously obtained, the connection is no longer valid. In this case, all connections currently in use by the application may prompt this exception.
- The application using the connection has already called close() and then tries to use the connection again.
- The connection has been orphaned, andthe application tries to use the orphaned connection.
- The application tries to use a JDBC resource, such as Statement, obtained on a now-stale connection.

When application code catches StaleConnectionException, it should take explicit steps to handle the exception. StaleConnectionException extends SQLException, so it can be thrown from any method that is declared to throw SQLException. The most common occasion for a StaleConnectionException to be thrown is the first time a connection is used, just after it has been retrieved. Because connections are pooled, a database failure is not detected until the operation immediately following its retrieval from the pool, which is the first time communication with the database is attempted. It is only when a failure is detected thatthe connection is marked stale. StaleConnectionException occurs less often if each method that accesses the database gets a new connection from the pool. Typically, this occurs because all connections currently allocatedto an application are marked stale; the more connections the application has, the more connections can be stale.

Generally,when a StaleConnectionException is caught, the transaction in which the connection was involvedneeds to be rolled back and a new transaction begun with a new connection.

For more information and detailed code samples,be sure to read the IBM whitepaper to be published on the Webduring the summer of 2001.

4.2.4.2.2: Accessing data with the JDBC 2.0 Core API

WebSphere applications can access a relational database directly through a JDBC provider that uses the JDBC 2.0 Core API.You access a relational database in this manner as follows:

1. Establish a connection through the DriverManager class
2. Send SQL queries or updates to the database management system
3. Process the results

Only a single connection is obtained. This connection does not belong to a pool of connections and is not managed by IBM WebSphere Application Server.It is the responsibility of the application to manage the use of this connection.

The following code fragment shows a simple example of obtaining and using a connection directly through a JDBC provider:

```
try {// establish a connection through the DriverManager
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");   String url = "jdbc:db2:sample";   String username
= "dbuser";   String password = "passwd";   java.sql.Connection conn =
java.sql.DriverManager.getConnection(url, username, password);// query the database
java.sql.Statement stmt = conn.createStatement();   java.sql.ResultSet rs =
stmt.executeQuery("SELECT EMPNO, FIRSTNME, LASTNAME FROM SAMPLE");    // process the results   while
(rs.next()) {      String empno = rs.getString("EMPNO");      String firstnme =
rs.getString("FIRSTNME");      String lastname = rs.getString("LASTNAME");   // work with results
}} catch (java.sql.SQLException sqle) {// handle the SQLException} finally {   try {      if(rs !=
null) rs.close();   }    catch (SQLException sqle) {   // can ignore   }   try {      if(stmt !=
null) stmt.close();   }    catch (SQLException sqle) {   // can ignore   }   try {      if(conn !=
null) conn.close();   }    catch (SQLException sqle) {   // can ignore   }}
```

In the previous example, the first action is to establish a connection to the database. This is done by loading and registering the JDBC provider andthen requesting a connection from DriverManager.DriverManager, a JDBC 1.0 class, is the basic service for managing a set of JDBC providers.It is necessary to load the driver class before the call to getConnection(), because DriverManager can establish a connection only to a driver that has registered with it. Loading the driver class also registers it with DriverManager.

After a connection has been obtained, the database is queried by creating a statement and executing a query on that statement. The query results are put into a ResultSet object.

Lastly, the results are processed by stepping through the result set and pulling the data from each record retrieved.

According to the JDBC 2.0 Core API specification, the DriverManager class has been deprecated. Therefore, any application using this class should be rewritten to use WebSphere connection pooling, which uses the datasource method described in the JDBC 2.0 Optional Package API to obtain connections to the database. For more information, see the JDBC 2.0 Core API specification.

4.2.4.2.3: Accessing relational databases with the IBM data access beans

Java programs that access JDBC-compliant relational databases typically use the classes and methods in the java.sql package to access data. Instead of using the java.sql package, you can use the classes and methods in the package com.ibm.db, the IBM data access beans. This gives you additional features for data access beyond those available in the java.sql package.

The Related information discusses what the data access beans are, their advantages, and how to use them. A data access bean uses a connection that you provide to it, such as a connection from a connection pool that you get through a DataSource object.

4.2.4.2.3.1: Example: Servlet using data access beans

The sample servlet uses the data access beans and is based on the sample servlet discussed in Article 4.2.4.2.1.1. The connection pooling sample servlet uses classes such as Connection, Statement, and ResultSet from the java.sql package to interact with the database. In contrast, this sample servlet uses the data access beans, instead of the classes in the java.sql package, to interact with the database. For convenience, call this sample servlet the DA (for data access beans) and call the sample servlet on which it is based the CP (for connection pooling).

The CP and DA sample servlets benefit from the performance and resource management enhancements made possible by connection pooling. The programmer coding the DA sample servlet benefits from the additional features and functions provided by the data access beans.

The DA sample servlet differs slightly from the CP sample servlet. This discussion covers only the changes. See Article 4.2.4.2.1.1 for the discussion of the CP sample servlet. The DA sample servlet shows the basics of connection pooling and the data access beans, but keeps other code to a minimum. Therefore, the servlet is not entirely realistic. You are expected to be familiar with basic servlet and JDBC coding.

The changes

This section describes how the DA sample servlet differs from the CP sample servlet. To view the coding in one or both of the samples while you read this section, click these links:

- DA sample
- CP sample

Steps 1 through 6 of the CP sample servlet are mostly unchanged in the DA sample servlet. The main changes to the DA sample servlet are:

- New package

  The com.ibm.db package (containing the data access beans classes) must be imported. The classes are in the databeans.jar file, found in the lib directory under the Application Server root install directory. You will need this jar file in your CLASSPATH in order to compile a servlet using the data access beans.
- The metaData variable

  This variable is declared in the Variables section at the start of the code, outside of all methods. This allows a single instance to be used by all incoming user requests. The full specification of the variable is completed in the init() method.
- The init() method

  New code has been appended to the init() method to do a one-time initialization on the metaData object when the servlet is first loaded. The new code begins by creating the base query object sqlQuery as a String object. Note the two "?" parameter placeholders. The sqlQuery object specifies the base query within the metaData object. Finally, the metaData object is provided higher levels of data (metadata), in addition to the base query, that will help with running the query and working with the results. The code sample shows:

    - The addParameter() method notes that when running the query, the parameter idParm is supplied as a Java Integer datatype, for the convenience of the servlet, but that idParm should be converted (through the metaData object) to do a query on the SMALLINT relational datatype of the underlying relational data when running the query.

      A similar use of the addParameter() method for the deptParm parameter notes that for the same underlying SMALLINT relational datatype, the second parameter will exist as a different Java datatype within the servlet - as a String rather than as an Integer. Thus parameters can be Java datatypes convenient for the Java application and can automatically be converted by the metaData object to be consistent with the required relational datatype when the query is run.

Note that the "?" parameter placeholders in the sqlQuery object and the addParameter() methods are related. The first addParameter() attaches idParm to the first "?", and so on. Later, a setParameter() will use idParm as an argument to replace the first "?" in the sqlQuery object with an actual value.

- ❍ The addColumn() method performs a function somewhat similar to the addParameter() method. For each column of data to be retrieved from the relational table, the addColumn() method maps a relational datatype to the Java datatype most convenient for use within the Java application. The mapping is used when reading data out of the result cache and when making changes to the cache (and then to the underlying relational table).
- ❍ The addTable() method explicitly specifies the underlying relational table. This information is needed if changes to the result cache are to be propagated to the underlying relational table.

- Step 5

Step 5 has been rewritten to use the data access beans to do the SQL query instead of the classes in the java.sql package. The query is run using the selectStatement object, which is a SelectStatement data access bean.

Step 5 is part of the process of responding to the user request. When steps 1 through 4 have run, the conn Connection object from the connection pool is available for use. The code shows:

1. The dataAccessConn object (a DatabaseConnection bean) is created to establish the link between the data access beans and the database connection - the conn object.
2. The selectStatement object (a SelectStatement bean) is created, pointing to the database connection through the dataAccessConn object, and pointing to the query through the metaData object.
3. The query is "completed" by specifying the parameters using the setParameter() method. The "?" placeholders in the sqlQuery string are replaced with the parameter values specified.
4. The query is executed using the execute() method.
5. The result object (a SelectResult bean) is a cache containing the results of the query, created using the getResult() method.
6. The data access beans offer a rich set of features for working with the result cache - at this point the code shows how the first row of the result cache (and the underlying relational table) can be updated using standard Java coding, without the need for SQL syntax.
7. The close() method on the result cache breaks the link between the result cache and the underlying relational table, but the data in the result cache is still available for local access within the servlet. After the close(), the database connection is unnecessary. Step 6 (which is unchanged from the CP sample servlet) closes the database connection (in reality, the connection remains open but is returned to the connection pool for use by another servlet request).

- Step 7

Step 7 has been entirely rewritten (with respect to the CP sample servlet) to use the query result cache retrieved in Step 5 to prepare a response to the user. The query result cache is a SelectResult data access bean.

Although the result cache is no longer linked to the underlying relational table, the cache can still be accessed for local processing. In this step, the response is prepared and sent back to the user. The code shows the following:

- ❍ The nextRow() and previousRow() methods are used to navigate through the result cache.

Additional navigation methods are available.

  - The getColumnValue() method is used to retrieve data from the result cache. Because of properties set earlier in creating the metaData object, the data can be easily cast to formats convenient for the needs of the servlet.

A possible simplification

If you do not need to updatethe relational table, you can simplify the sample servlet:

- At the end of the init() method, you can drop the lines with the addColumn() and addTable() methods, since the metaData object does not need to know as much if there are no relational table updates.
- You will also need to drop the lines with the setColumnValue() and updateRow() methods at the end of step 5, because you are no longer updating the relational table.
- Finally, you can remove most of the type casts associated with the getColumnValue() methods in step 7. You will, however, need to change the type cast to (Short) for the "ID" and "DEPT" use of the getColumnValue() method.

4.2.4.2.4: Database access by servlets and JSP files

Servlets using getConnection() to access a data source

When used without parameters, getConnection() assumes the default user ID and password for a data source. The WebSphere administrative clients do not offer a way to configure a default user ID and password for a data source to be used by a servlet.

Therefore, servlets using getConnection() to access a data source should specify a user ID and password:

```
getConnection(userid,password);
```

6.6.14: Administering database connections (overview)

The administrator of Advanced Edition (non-Single Server Edition) establishesdatabase connections for supporting data access by:

- Applications installed in the server runtime, possibly including the WebSphere Samples
- The WebSphere administrative server for obtaining its administrative data

For information about the latter topic, see the article about administeringthe administrative server and database.

The WebSphere administrator has an important role in establishingand maintaining connection pools through data sources and drivers:

- Configuring the resources used in connection pooling -- JDBC drivers and data sources

  The administrator needs to configure data sources and JDBC drivers for each brand and version of database from which enterprise applications or individual resources will require connections.
- Adjusting connection pooling parameters for optimal performance.

  Connection pools provide a way to share the connection overhead among multiple requests, but it is possible to configure too large a connection pool. The administrator needs to determine the optimal value for the pool size and other settings, based on environmental factors such as the operating system in use.

JDBC Drivers and data sources are used by Java components requiringdatabase access, such asservlets and enterprise beans. All entity beans require a datasource, either defined as a property of the enterprise bean, or as a property of the enterprise bean container.

6.6.14.0: Properties of JDBC and data source providers

The administrative tools use the following terms for the same type of configuration:
JDBC provider or data source provider

Key:

Applies to Java administrative console of Advanced Edition Version 4.0

Applies to Web administrative console of Advanced Single Server Edition Version 4.0

Applies to Application Client Resource Configuration Tool

Classpath or Server Class Path

The path to the JAR file containing the implementation code for the database driver, such as db2java.zip file for DB2.

See the reference below for a list of default locations.

Custom Properties

Name-value pairs for setting additional properties

Description

A description of the driver, for your administrative records

Implementation Class or Implementation Classname

The name of the Java data source class provided by the database vendor.

DB2:

- COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource (one phase commit protocol)
- COM.ibm.db2.jdbc.DB2XADataSource (two phase commit protocol)

DB2 on iSeries - toolbox driver

- com.ibm.as400.access.AS400JDBCConnectionPoolDataSource (one phase commit protocol)
- com.ibm.as400.access.AS400JDBCXADataSource (two phase commit protocol)

DB2 on iSeries - native driver

- com.ibm.db2.jdbc.app.DB2StdConnectionPoolDataSource (one phase commit protocol)
- com.ibm.db2.jdbc.app.DB2StdXADataSource (two phase commit protocol)

Oracle:

- oracle.jdbc.pool.OracleConnectionPoolDataSource (one phase commit protocol)
- oracle.jdbc.xa.client.OracleXADataSource (two phase commit protocol)

Sybase:

- com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource (one phase commit protocol)
- com.sybase.jdbc2.jdbc.SybXADataSource (two phase commit protocol)

Merant:

  - com.merant.sequelink.jdbcx.datasource.SequeLinkDataSource (one and two phase commit protocol)

Informix

  - com.informix.jdbcx.IfxConnection PoolDataSource (one phase commit protocol)
  - com.informix.jdbcx.IfxXADataSource (two phase commit protocol)

Name

A name for the driver. It is recommended you enter a name that is suggestive of the database product you are using, such as DB2JdbcDriver.

Node

The node (machine) on which to install the driver.

For Advanced Edition: Use the buttons on the Nodes panel to access dialogs for installing drivers on specific nodes and for uninstalling drivers.

Property Set

See the resource provider properties (only valid if you are using *Advanced Single Server Edition*).

Server Class Path

See Classpath

URL prefix

The URL prefix with which this driver is associated. The URL prefix is comprised of the protocol and subprotocol, separated by a colon (":").

The Database Name of the data source is appended to the URL prefix to produce the full JDBC URL of the database, such as jdbc:db2:was

Locating JDBC providers on your operating system

The following table lists the default locations of JDBC provider files. Seethe product prerequisites Web site for the most up-to-date information on the operating system brands and databases supported by IBM WebSphere Application Server. Column 1 in the table lists the operating system, and column 2 shows a list of the drivers that are available for use with each database supported for the operating system.

Operating system	Drivers
AIX	● DB2: *$DB2_HOME*/java12/db2java.zip or *$DB2_HOME*/java/db2java.zip ● Oracle: *$ORACLE_HOME*/jdbc/lib/classes12.zip ● Sybase: *sybase_install_root*/jConnect-5_2/classes/jconn2.jar Merant SequelLink: ❍ Complimentary copies with WebSphere Application Server: ■ *product_installation_root*/lib/sljc.jar ■ *product_installation_root*/lib/sljcx.jar

HP-UX	● DB2: *$DB2_HOME*/java12/db2java.zip or *$DB2_HOME*/java/db2java.zip ● Oracle: *$ORACLE_HOME*/jdbc/lib/classes12.zip ● Merant SequelLink: ❍ Complimentary copies with WebSphere Application Server: ■ *product_installation_root*/lib/sljc.jar ■ *product_installation_root*/lib/sljcx.jar
Linux (Intel)	● DB2: *$DB2_HOME*/java12/db2java.zip or *$DB2_HOME*/java/db2java.zip ● Oracle: *$ORACLE_HOME*/jdbc/lib/classes12.zip
Linux on S/390	● Oracle: *$ORACLE_HOME*/jdbc/lib/classes12.zip
Solaris	● DB2: *$DB2_HOME*/java12/db2java.zip or *$DB2_HOME*/java/db2java.zip ● Oracle: *$ORACLE_HOME*/jdbc/lib/classes12.zip ● Sybase: *sybase_install_root*/jConnect-5_2/classes/jconn2.jar Merant SequelLink: ❍ Complimentary copies with WebSphere Application Server: ■ *product_installation_root*/lib/sljc.jar ■ *product_installation_root*/lib/sljcx.jar
Windows	● DB2: C:\SQLLIB\java\db2java.zip ● Oracle: C:\Oracle\Ora81\jdbc\lib\classes12.zip ● Merant SequelLink: ❍ Complimentary copies with WebSphere Application Server: ■ *product_installation_root*\lib\sljc.jar ■ *product_installation_root*\lib\sljcx.jar

6.6.14.0.1: Properties of data sources

Key:

Applies to Java administrative console of Advanced Edition Version 4.0

Applies to Web administrative console of Advanced Single Server Edition Version 4.0

Applies to Application Client Resource Configuration Tool

Category

An optional category string that can be used to classify or group the resource

Confirm Password **or Re-Enter Password**

Confirm the password that you entered in the preceding field

Connection timeout

The maximum time in seconds that requests for a connection wait if the maximum number of connections is reached and all connections are in use

This value must be a positive integer.

Custom Properties **or Property Set**

A set of custom name-value pairs describing properties of the data source

The following are the <u>required</u> properties for each database type:

DB2:

No required properties.

DB2 on iSeries -- toolbox driver:

serverName

The name of the server from which the data source will obtain connections, such as "MyServer"

DB2 on iSeries -- native driver:

No required properties.

Oracle:

URL

The url indicating the database from which the data source will obtain connections, such as "jdbc:oracle:thin:@myServer:1521:myDatabase," where "myServer" is the server name, "1521" is the port it is using for communication, and "myDatabase" is the database name.

Set the user and password in the field provided in the console.

Sybase:

serverName

The name of the database server, such as "db_machine" portNumber

The tcpip port number through which all communications to the server take place, such as 4100.

Merant:

serverName

The name of the server where SequeLinkServer resides, such as "MyServer"

portNumber

The TCP/IP port SequeLinkServer uses for communication. By default, SequeLinkServer uses port 19996, such as "19996".

disable2Phase

By default, two phase connections ae used by Merant always, because the same data source class is used for one phase and two phase commit protocols.

To use one phase connections, set disable2Phase to true.

Set the user and password in the field provided in the console.

Informix:

ifxIFXHOST

The physical machine name

serverName

The name of the Informix instance on the physical machine

portNumber

The port number of the Informix instance

informixLockModeWait

By default, Informix throws an exception when it cannot acquire a lock, rather than waiting for the current owner of the lock to release it. To modify this behavior, set this property to the number of seconds to wait for a lock. The default is 0 seconds. Any negative value means to wait forever.

The following are some additional, optional properties for various database types:

Sybase:

connectionProperties

CHARSET_CONVERTER_CLASS=com.sybase.jdbc2.utils.TruncationConverter

Setting the CHARSET_CONVERTER_CLASS can prevent exceptions such as this one when performing a dataSource.getConnection() call:

```
java.io.IOException: JZ0I6: An error occured converting
UNICODE to the charset used by the server. Error
message: java.io.CharConversionException:
java.io.UnsupportedEncodingException: hp-roman8
```

Set additional connectionProperties by specifying them using the same pattern, separated by commas: *PROPERTY_NAME*=value;*PROPERTY_NAME*=value; ...

Database Name

The name of the database used to store entity bean data

This is required for DB2, and sometimes required for Sybase, Merant, and Informix (depending on your database configuration), and ignored for Oracle.

Description

A description of the data source, for your administrative records

Default Password **or Password**

The password for connecting to the database when no user ID and password pair is specified by the application. If the default password is specified, the default user ID must also be specified.

Default User ID **or User**

The user name for connecting to the database when no user ID and password pair is specified by the application. If the default user ID is specified, the default password must also be specified.

Disable Auto Connection Cleanup

Keeps the connection pooling software from automatically closing connections from this data source at the end of a transaction. This behavior is needed if you want to reuse the same connection across multiple transactions. When this is set, you must be sure to close the connection programmatically when you are through using it.

Idle timeout

The maximum time in seconds that an idle (unallocated) connection can remain in the pool before being removed to free resources.

This value must be a positive integer.

JDBC Provider

The JDBC provider (also known as data source provider) with which this data source is associated. It is used to connect to a relational database.

JNDI Name

The JNDI name for the resource, including any naming subcontexts. This name is used as the linkage between the platform's binding information for resources defined in the client application's deployment descriptor and actual resources bound into JNDI by the platform.

Maximum Connection Pool Size **or Maximum Pool Size**

The maximum number of connections that can be in the pool. If the maximum number of connections is reached and all connections are in use, additional requests for a connection wait up to the number of seconds specified in the Connection timeout property.

This value must be a positive integer.

Maximum Pool Size

See Maximum Connection Pool Size

Minimum Connection Pool Size **or Minimum Pool Size**

The minimum number of connections in the pool.

This value must be a positive integer.

Minimum Pool Size

See Minimum Connection Pool Size

Name

A name by which to administer the data source.

It is recommended that you enter a name that is suggestive of the database you will use to store entity bean data, such as WASDataSource, where WAS is the database name. The default value for this property is the value of the Name property prefixed with "jdbc/" (such as "jdbc/DataSourceName").

Orphan timeout

The maximum number of seconds that an application can hold a connection without using it before the connection can be returned to the pool.

This value must be a positive integer.

Note that the actual amount of time before a connection is closed is approximately twice the orphan timeout value.

Password

See Default Password

Property Set

See the custom properties

Re-Enter Password

See Confirm Password

Statement Cache **or Statement Cache Size**

The maximum number of prepared statements to cache for the data source. The limit is shared among all connections. The default value is 100.

User

See Default User ID

6.6.14.1: Administering database connections with the Java console

Use the Java administrative console to administer JDBC providers and data sources.

Work with resources of this type by locating them in the tree view:

WebSphere Administrative Domain -> Resources -> JDBC Providers -> ***provider_name***

Expand the tree further to see **Data Sources ->** ***datasource_name*** for each provider.

Data source instances will only be visible if they have been created.

6.6.14.1.1: Configuring new database connections with the Java administrative console

To configure JDBC providers and data sources, the resources necessary to allow applications to connect to databases, click **Console -> Wizards -> Create Data Source** from the console menu bar.This leads to the Datasource task wizard.

Using the Datasource wizard

You use the Datasource wizard to define and configure connections toa relational database that supplies a JDBC provider. Using the wizard sets some properties pertaining to data sources. To set all values for all of the properties, use theproperties dialogs for data access.

- Naming the data source
- Selecting whether to use an existing or new JDBC provider
- Creating a new JDBC provider
- Completing the data source

Naming the data source

On the Specifying Datasource Resources panel:

1. Name your data source.

   It is recommended that within the name you identify the database to which the data source will provide connections. For example, enter `WASDatasource`, where *WAS* is the name of the database. The name of the JNDI lookup for such a data source would be *jdbc/WASDatasource*.
2. Describe your data source.
3. If required by your database, specify the name of the database being connected to.

   The name is required for DB2, and sometimes required for Sybase, Merant, and Informix (depending on your database configuration), and ignored for Oracle.

   For example, for DB2, you could enter `WAS`, and your data source would, as a result, point to *jdbc:db2:WAS*. For Oracle, you would leave this field blank.

   View data source property reference
4. Click **Next**.

Selecting whether to use an existing or new JDBC provider

On the Specifying a JDBC Provider panel:

1. If your data source will use a JDBC provider previously configured, select the provider from the drop-down list for **Use an existing JDBC provider**. Otherwise, opt to create a new JDBC provider.
2. Click **Next**.

Creating a new JDBC provider

If you opted to create a new JDBC provider, you will see the Creating a JDBC Provider panel:

1. Name the provider. It is recommended that the name identify the database using the JDBC provider. For example, enter `DB2JdbcDriver`.
2. Describe the provider.
3. Select the Java class corresponding to the database being used. click the ellipses to open a dialog in which you can select the implementation class or enter an implementation class in the text field of the ellipses field.

   View data access properties help
4. Click **Next**.

Completing the data source

The Completing the Datasource Wizard panel lists the database and provider names, and the implementation class.

If you do not want to change the values specified, click **Finish**. Thewizard will define a data source with the values you specified, and display a message indicating whether the data source was successfully created. If the wizard encounters an error, a message will display explaining why a data sourcecould not be defined.

To change the values specified, click **Back** to return to the appropriatepanel(s), make any needed changes, and then click **Finish** on this panel.

⚠ For many databases, additional propertiesneed to be added after the Datasource Wizard is completed, before the data sourcecan be used. For a list of the properties required for the data source for eachdatabase type, see the **Custom Properties** field description in the data source property reference.

6.6.14.5: Additional administrative tasks for specific databases

For your convenience, this article provides instructions for enabling some popular database drivers, and performing other administrative tasks often required in order to provide dataaccess to applications running on WebSphere Application Server. These tasks are performedoutside of the WebSphere Application Server administrative tools, often using the databaseproduct tools. Always refer to the documentation accompanying your database driver as the authoritative andcomplete source of driver information.

See the WebSphere Application Server product prerequisites for the latest information about supported databases, drivers, and operating systems.

- Enabling JDBC 2.0
  - For DB2 on Windows NT
  - For DB2 on UNIX
- Sourcing the db2profile script on UNIX
- Using JTA drivers
  - For DB2 on Windows NT
  - For DB2 on UNIX
- For Oracle 8.1.7 two phase commit support
- For Sybase on AIX

Enabling JDBC 2.0

Ensure that your operating system environment is set up to enable JDBC 2.0 use.This is required in order to use data sources created through WebSphere ApplicationServer.

The following steps make it possible to find the appropriate JDBC 2.0 driverfor use with WebSphere Application Server administration:

Enabling JDBC 2.0 with DB2 on Windows NT

To enable JDBC 2.0 use on Windows NT systems:

1. Stop the DB2 JDBC Applet Server service.
2. Run the following batch file:
   ```
   SQLLIB\java12\usejdbc2.bat
   ```
3. Stop WebSphere Application Server (if it is running) andstart it again.

Perform the steps once for each system.

Determining the level of JDBC in use for DB2 on Windows NT

To see whether the JDBC level in use on your system:

- If JDBC 2.0 is in use, this file will exist:
  ```
  SQLLIB\java12\inuse
  ```
- If JDBC 1.0 is in use, this file will exist:
  ```
  SQLLIB\java11\inuse
  ```
  or there will be no java11 directory

Enabling JDBC 2.0 with DB2 on UNIX

Before starting WebSphere Application Server, you need to call $INSTHOME/sqllib/java12/usejdbc2 to use JDBC 2.0. For convenience, you might want to put this in your root user's startup script.For example on AIX, add the following to the root user's .profile:

```
if [ -f /usr/lpp/db2_07_01/java12/usejdbc2 ] ; then    . /usr/lpp/db2_07_01/java12/usejdbc2fi
```

Determining the level of JDBC in use for DB2 on your UNIX system

To determine if you are using JDBC 2.0, you can echo $CLASSPATH. If it contains

```
$INSTHOME/sqllib/java12/db2java.zip
```

then JDBC 2.0 is in use.

If it contains

```
$INSTHOME/sqllib/java/db2java.zip
```

then JDBC 1.0 is in use.

Sourcing the db2profile script on UNIX

Before starting WebSphere Application Server to host applications requiringdata access, source the db2profile:

```
.~db2inst1/sqllib/db2profile
```

where *db2inst1* is the user created during DB2 installation.

Using JTA drivers

Instructions are available for using JTA drivers on particular operatingsystems. See your operating system's documentation for more information.

With the JDBC programming model underlying WebSphere Application Server Version 4.0x, the term "JTA enabled" becomes somewhat odd, with respect toits Version 3.5 meaning. The goal of this section about "Using JTA drivers" is to provide information about the steps that make DB2 work nicely with applicationsutilizing XA classes -- that is, those whose dataSourceClasses implement javax.sql.XADataSource.

Using JTA drivers for DB2 on Windows NT

To enable JTA drivers for DB2 on Windows NT, follow these steps:

1. Stop all DB2 services.
2. Stop the IBM WebSphere Application Server administrative service.
3. Stop any other processes that use the db2java.zip file. (Note: If the **JVIEW** process is active, you must use the **Task Manager** utility to stop it.)
4. Make sure that you already enabled JDBC 2.0.
5. Start the DB2 services.
6. Configure DB2 to use JTS as the transaction processing (TP) monitor. From the DB2 Control Center, follow these steps:
   a. Right-click the DB2 instance that contains the database that is to be enabled for JTA access.
   b. Click **Multisite Update**, **Configure** to start the Smartguide utility.
   c. Click the **Use the TP monitor named below** radio button.
   d. Select **JTS** as the TP monitor.
   e. Click **Done**.
7. Bind the necessary packages to the database. From the **DB2 Command Line Processor** window, issue the following commands:

   ```
   db2=> connect to mydb2jtadb2=> bind db2home\bnd\@db2cli.lstdb2=> bind db2home\bnd\@db2ubind.lstdb2=>
   disconnect mydb2jta
   ```

   where *mydb2jta* is the name of the database that is to be JTA enabled, and *db2home* is the DB2 root installation directory path (for example, D:\ProgramFiles\SQLLIB\bnd\@db2cli.lst).
8. When you use an IBM WebSphere Application Server administrative client (suchas the WebSphere Administrative Console) to configure a JDBC provider, specifythe following settings:
   - **Server class path** = `%DB2_ROOT%/Sqllib/java/db2java.zip`
   - **Implementation class name** = `COM.ibm.db2.jdbc.DB2XADataSource`

Using JTA drivers for DB2 on UNIX

To enable JTA drivers on UNIX, follow these steps:

1. Stop all DB2 services.
2. Stop the IBM WebSphere Application Server administrative service.
3. Stop any other processes that use db2java.zip file.
4. Make sure that you already enabled JDBC 2.0.
5. Start the DB2 services.
6. Bind the necessary packages to the database.From the DB2 **Command Line Processor** window,issue the following commands:

   ```
   db2=> connect to mydb2jtadb2=> bind db2home\bnd\@db2cli.lstdb2=> bind db2home\bnd\@db2ubind.lstdb2=>
   disconnect mydb2jta
   ```
7. When you use an IBM WebSphere Application Server administrative client (suchas the WebSphere Administrative Console) to configure a JDBC provider, specifythe following settings:
   - **Server class path** = `$INSTHOME/sqllib/java12/db2java.zip`

     For example, if $INSTHOME is `/home/test`, the path will be `/home/test/sqllib/java12/db2java.zip`
   - **Implementation class name** = `COM.ibm.db2.jdbc.DB2XADataSource`

For Oracle 8.1.7 two phase commit support

The Oracle 8.1.7 thin driver can be used for JTA two phase support with the following restrictions:

- The thin driver that comes shipped with 8.1.7 may or may not work. Futurepatches from Oracle may work as well, but have not been tested. The driver that was available from the Oracle Technology Network Web site as of February 20, 2001 does work and is the recommended driver. Later versions on this Web site are expected to work, but have not been tested.

  To obtain the driver from the Oracle support Web, visit:

  ```
  http://technet.oracle.com/
  ```

You will need to be a registered user for the Oracle Technology Network to get the driverfrom this site. Contact Oracle for how to get access. After you have access:

1. On the left hand side of the screen, select "Software".
2. On "Download Oracle Products, Drivers, and Utilities"
3. On the "Select a Utility or Driver" selection, select "Oracle JDBC Providers".
4. Download the 8.1.7 driver for the platforms you use and follow the instructions for installing the new driver.

The above instructions are subject to change with no notice to IBM. This versionof the instructions could become inaccurate. Consult the Oracle site for the definitiveinstructions.

- The 8.1.7 driver must be used with 8.1.7 databases. 8.1.6 databases do notsupport the recover() and forget() methods and other problems have been encountered runningwith 8.1.6. Oracle does not support JTA with 8.1.6.
- For Oracle, JTA can only be used with container managed (CMP) beans.
- In order for the bean to create the table, the bean must first be started the JTA set tofalse. After the bean has created the table, JTA can be set back to true.
- An entity bean that accesses Oracle with JTA set to true must be configuredas follows:
  - In the deployment descriptor properties, under Transactions, under the Remote tab, set the Trasnaction Attribute to TX_REQUIRED.
  - Under Isolation, under the Remote tab, set the Isolation Level to TRANSACTION_READ_COMMITTED.
- A session bean that is used with an entity bean that accesses Oracle with JTA set totrue must be configured as follows:
  - In the deployment descriptor properties, Under Transactions, under the Remote tab, set the Transaction Attribute to TX_BEAN_MANAGED.
  - Under Isolation, under the Remote tab, set the Isolation Level to TRANSACTION_READ_COMMITTED.

Using JTA drivers for Sybase on AIX

To enable JTA drivers for use with Sybase on the AIX operating system,follow these steps:

1. At a command prompt, enable the Data Transaction Manager (DTM) by issuing these commands (one per line):

   ```
   isql -Usa -Ppassword -Sservername       sp_configure "enable DTM", 1    go
   ```

2. Stop the Sybase Adaptive Server database and start it again.
3. At a command prompt, grant the appropriate role authorization to the EJB user:

   ```
   isql -Usa -Ppassword -Sservername       grant role dtm_tm_role to EJB                   go
   ```

Notes about Sybase JTA drivers

Do not use a Sybase JTA connection in an enterprise bean method with an unspecified transaction context. A Sybase JTA connection does not support the local transaction mode. The implication is that the Sybase JTA connection must be used in a global transaction context.

6.6.14.6: Notes about various databases

This article provides miscellaneous tips for using supported databases. See also the related links.

Alwaysconsult the product prerequisites Web site for a list of the database brands and versions that aresupported by your particular Application Server version, edition, and fix pack.

- Do not drop the administrative database while the administrative server is running.
- DB2 performance on a local machine can be improved by setting up a local database as a remote instance. On UNIX systems, this is required. The configuration uses TCP/IP instead of shared memory.

  Oracle and Sybase also support client/server connections. Consult their product documentation for specifics.
- If using local DB2 databases for data access by session clients, in some cases, multiple connections for session clients cannot be established successfully. To avoid stale connections when there are large numbersof session clients, catalog the DB2 databases using a TCP/IP loopback.
  1. Set up a TCP/IP port in /etc/services, if a port for remote DB2 clients has not been established yet.
  2. Ensure that the TCP/IP communication protocol has been specified in the DB2COMM registry parameter.
     - To check the current setting of the DB2COMM parameter, enter `db2set DB2COMM`.
     - To update the DB2COMM registry variable to include TCP/IP, use the db2set command.

     For example:

     ```
     db2set DB2COMM=existing_protocol_names, tcpip
     ```
  3. Update the SVCENAME database manager configuration parameter to the connection service name as defined in /etc/services (step 1). For example:

     ```
     db2 update dbm cfg using svcename connection service name
     ```
  4. Catalog the loopback node. For example:

     ```
     db2 catalog tcpip node node_name remote 127.0.0.1 server connection_service_name
     ```
  5. Catalog the database as follows:

     ```
             db2 catalog db database_name as database_alias        db2 uncatalog db database_name
     db2 catalog db database_alias as database_name at node node_name
     ```

     This allows you to implement a TCP/IP loopback without needing to change the application to connect to the new alias and the USER and USING parameters.
  6. Stop DB2 and start it again to refresh the directory cache.
- When using Sybase 11.x, you might encounter the following error when HttpSession persistence is enabled:

  ```
  DBPortability W Could not create database table: "sessions" com.sybase.jdbc2.jdbc.SybSQLException:
  The 'CREATE TABLE' command is notallowed within a multi-statement transaction in the 'database_name'
  database
  ```

  where 'database_name' is the name of the database for holding sessions.

  If you encounter the error, issue the following commands at the Sybase command line:

  ```
  use database_namegosp_dboption db,"ddl in tran ",truego
  ```
- Sybase 12.0 does not support local transaction modes with a JTA enabled data source. To usea connection from a JTA enabled data source in a local transaction, Sybase patch EBF9422 mustbe installed.

6.6.14.8: Recreating database tables from the exported table DDL

To recreate database tables from the exported table DDL, you execute the exported table DDL file to create a table for CMP beans by hand. For instructionsfor exporting the table DDL, see the following:

- Exporting DDLs of EJB modules

Create the tables in the database of the data source configuration in the following order:

1. If you specified data source for the CMP bean, then use the data source for the bean.
2. If you specified data source for the EJB module of that CMP bean, then use the data source for the EJB module.
3. If neither of the above two conditions applies, use the default data source for the EJBContainer of the application server onto which the EJM module was installed.

The content of the table DDL file is the SQL strings needed to create a table. The content differs for different databases. The DDL file is essentially an SQL file, and each database has a different command to execute an SQL file.

Copy or paste the content of the table DDL file to a database command line, or use the command line option to take a SQL file as a parameter. The syntax of the command for supported databases is:

For DB2:

```
db2 -tf table_DDL_file_name
```

For Oracle:

```
sqlplus user_name/password
```

After a new window displays:

```
@table_DDL_file_name
```

For Sybase:

```
isql -Uuser_name -Ppassword -Sserver_name -i table_DDL_file_name
```

6.6.14.9: Administering data source providers and data sources with the Application Client Resource Configuration Tool

Use the Application Client Resource Configuration Tool to edit the configurations of data source providers (such as JDBC providers)and data sources, which are used by your application clients to access data fromdatabases.

Work with objects of this type by locating them in the tree that is displayed by the tool when you use it to openan EAR file. If file with which you are working contains data source providers and data sources, its tree will contain one or more of the following:

Resources -> ***application*****.jar** -> **Data Source Providers** -> ***data_source_provider_instance***

where ***data_source_provider_instance*** isa particular data source provider.

If you expand the tree further, you will also see the **Data Sources** folders containing the data source instances for each data source provider instance.

6.6.14.9.1: Configuring new data source providers (JDBC drivers) with the Application Client Resource Configuration Tool

During this task, you will create new data source providers (also known as JDBC providers)for your client application. Note, in a separate administrative task, the Java code for the required data source provider must be installed on the client machine on which the client application resides.

To configure a new data source provider:

1. Start the tool and open the EAR file for which you want to configure the new data source provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file in which you wantto configure the new data source provider.
3. Expand the JAR file to view its contents.
4. Click the folder called **Data Source Providers**. Do one of the following:
    - Right-click the folder and select **New Provider**.
    - On the menu bar, click **Edit** -> **New**.
5. In the resulting property dialog, configure the data source provider properties.
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.14.9.1.1: Configuring new data sources with the Application Client Resource Configuration Tool

During this task, you will create new data sources for your client application.

1. In the tree, click the data source provider for which you want to create a data source.
   - Configure a new data source provider.
   - Or, click an existing data source provider.
2. Expand the data source provider to view its **Data Sources** folder.
3. Click the folder. Do one of the following:
   - Right-click the folder and select **New Factory**.
   - On the menu bar, click **Edit** -> **New**.
4. In the resulting property dialog, configure the data source properties.
5. When finished, click **OK**.
6. On the menu bar, click **File** -> **Save** to save your changes.

6.6.14.9.3: Removing data source providers (JDBC drivers) and data sources with the Application Client Resource Configuration Tool

Please see "Removing objects from EAR files with theApplication Client Resource Configuration Tool", as this task is similar for all object types supported by the tool.

6.6.14.9.4: Updating data source and data source provider configurations with the Application Client Resource Configuration Tool

During this task, you will modify (update) the configuration of an existing data source ordata source provider.

1. Start the tool and open the EAR file containing the data source or data source provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file containing the data source or data source providerthat you want to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular data source or datasource provider that you want to update. When you find it, do one of the following:
    - Right-click the object and select **Properties**
    - On the menu bar, click **Edit** -> **Properties**
5. In the resulting property dialog, update the properties. For detailed field help, see:
    - Data source provider properties
    - Data source properties
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.