Working with Database Tables, DataSources and JMS Resources

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Working With Database Tables and DataSources

Purpose

The purpose of this guide is to introduce you to the basics of working with database tables and deploying custom DataSources for your applications developed with the SAP NetWeaver Application Server, Java(TM) EE 5 Edition.

It starts by introducing the preconfigured tables and DataSource that you might use out-of-the-box and then goes on to elaborate further on how you can create your own tables and DataSources using the tools provided in the SAP NetWeaver Developer Studio.

Integration

The best way to explore the capabilities of the SAP NetWeaver Application Server, Java(TM) EE 5 Edition is, of course, to try and write your own applications using the samples that come with the SAP NetWeaver Developer Studio as a basis. Each of those samples comes with its own database content, so you could easily use the very same tables that we created for the persistence of your own applications. To give you an overview of the predefined content of the database, here is the script that we used to create some of them: demotables.sql [Page 8].

This script created the tables in a special database schema for the newly-created user SAPDEMO, whose password is also SAPDEMO. Furthermore, it references the database instance JP1, which is used as the default in the SAP NetWeaver Application Server, Java(TM) EE 5 Edition. You can use the script as an example in case you have to define new tables for your own applications.

Creating Your Own Tables in the MaxDB Database

Creating Tables Using SQLCLI Script

If the set of predefined tables is not sufficient for your needs, you can simply add more tables by copying the script from the last section into a file and then modifying it. You then run the script with the command line tool SQLCLI. For example, the following dialog would run the script demotables.sql for user SUPERDBA with password `mymaster`.

`mymaster` should be the master password supplied during installation. It is case-sensitive.

```
N:\>sqlcli -d JP1 -u superdba,mymaster
Welcome to the MaxDB interactive terminal.
sqlcli=> \i C:\myscripts\demotables.sql
sqlcli=> \q
```

Of course, SQLCLI offers much more functionality for entering and executing SQL statements, executing database procedures, and querying information about the database instance. For a complete list of commands, consult the “Tools” section in the online documentation for the MaxDB at **`dev.mysql.com`**.

You can use the command line tool DBMCLI for executing virtually all MaxDB administrative tasks. For example, the following line can be used to find out whether the MaxDB is running and online:

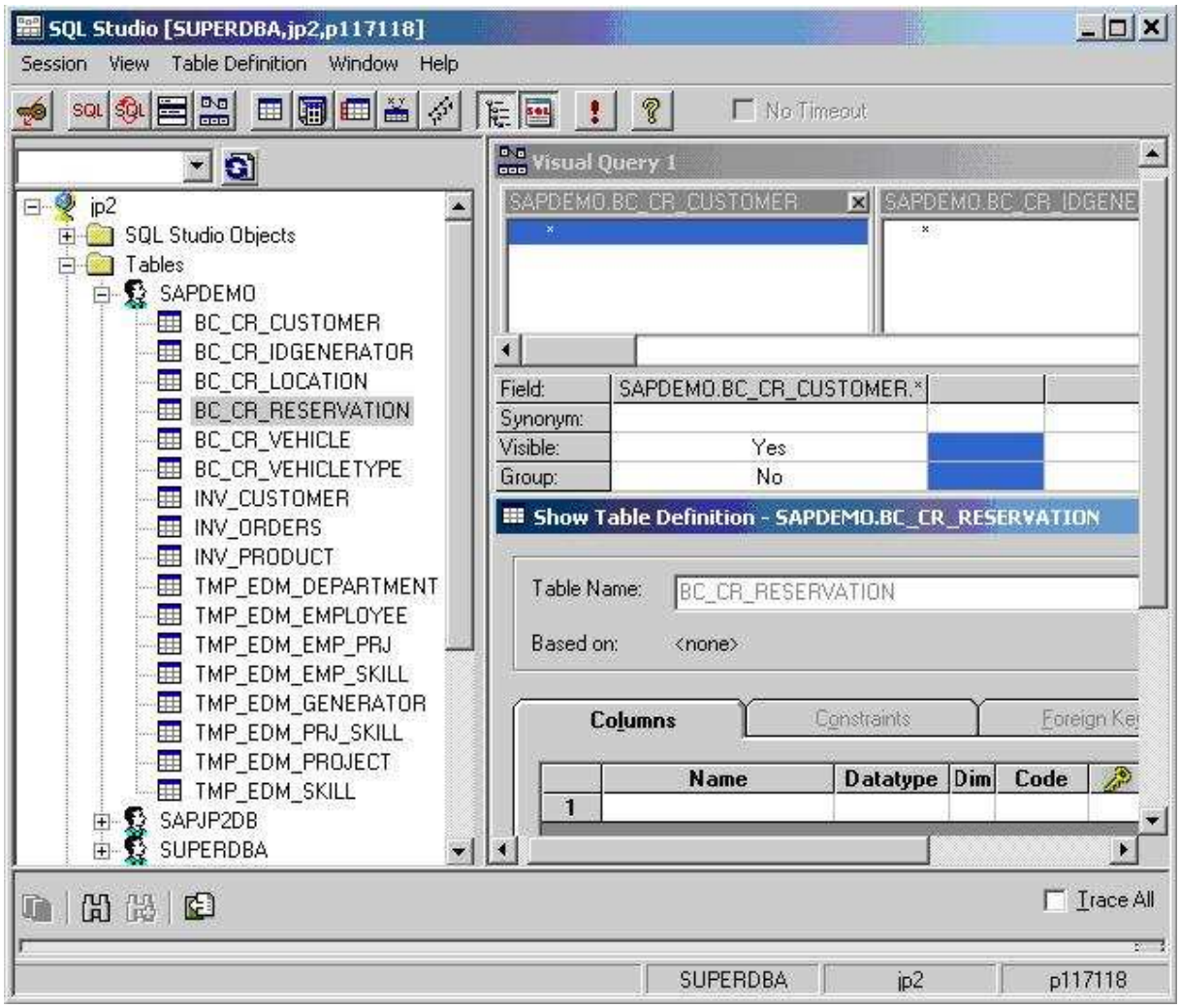
```
N:\>dbmcli -d JP1 -u control,mymaster db_state
OK
State
ONLINE
```

Again, the “Tools” section in the MaxDB online documentation has a full list of commands.

Creating Tables Using SQL Studio

The second and much more comfortable way of managing the content of the MaxDB is offered by the SQL Studio. The SQL Studio can be downloaded from **`dev.mysql.com`**.

With the SQL Studio you can not only create tables, but easily access all the application data as well as the database catalog of a MaxDB database instance. You can create, execute, and manage any number of SQL statements, and you can create, display, or change database catalog objects.



When connecting to the MaxDB instance (JP1) from the SQL Studio, the same users and passwords apply that were mentioned in the last paragraph. You can either use the predefined user SUPERDBA with the master password, or the user SAPDEMO with password SAPDEMO.

demotables.sql

```
// this line assumes that mymaster is the master password that you
// provided during installation

\connect -d JP1 -u SUPERDBA,mymaster

//

DROP USER SAPDEMO

//

CREATE USER SAPDEMO PASSWORD SAPDEMO DBA NOT EXCLUSIVE

//

\disconnect

//

\connect -u SAPDEMO,SAPDEMO

//------------------------------------------------------------------

// Create tables for Car Rental Application

//------------------------------------------------------------------


CREATE TABLE BC_CR_CUSTOMER
(
    CUSTOMERID     Varchar (10)  UNICODE    NOT NULL    DEFAULT '0',
    FIRSTNAME      Varchar (35)  UNICODE    NOT NULL    DEFAULT "",
    LASTNAME       Varchar (35)  UNICODE    NOT NULL    DEFAULT "",
    USERNAME       Varchar (50)  UNICODE    NOT NULL    DEFAULT 'user',
    PASSWORD       Varchar (10)  UNICODE    NOT NULL    DEFAULT "",
    FORM           Varchar (10)  UNICODE,
    COUNTRY        Varchar (3)   UNICODE    NOT NULL    DEFAULT "",
```

```
    REGION         Varchar (50)  UNICODE,
    CITY           Varchar (50)  UNICODE    NOT NULL    DEFAULT "",
    POSTALCODE     Varchar (10)  UNICODE    NOT NULL    DEFAULT '0',
    ADDRESS        Varchar (100) UNICODE    NOT NULL    DEFAULT "",
    DRIVERLICENSE  Varchar (15)  UNICODE    NOT NULL    DEFAULT '0',
    PASSPORTNR     Varchar (15)  UNICODE    NOT NULL    DEFAULT '0',
    PHONE          Varchar (20)  UNICODE,
    MOBILE         Varchar (20)  UNICODE,
    EMAIL          Varchar (75)  UNICODE,
    CREDITCARDID   Varchar (16)  UNICODE,
    CREDITCARDTYPE Varchar (20)  UNICODE,
    COMPANY        Varchar (50)  UNICODE,
    USERTYPE       Varchar (20)  UNICODE    NOT NULL    DEFAULT 'regular customer',
     CUMULATEDVALUE Fixed (16,2)   NOT NULL    DEFAULT            0.00,
    PRIMARY KEY (CUSTOMERID)
)

//

CREATE TABLE BC_CR_IDGENERATOR
(
    TABLENAME  Varchar (50) UNICODE    NOT NULL    DEFAULT "",
    LASTID     Varchar (10) UNICODE    NOT NULL    DEFAULT '0',
    PRIMARY KEY (TABLENAME)
)

//

CREATE TABLE BC_CR_LOCATION
(
    LOCATIONID   Varchar (10) UNICODE    NOT NULL    DEFAULT ' ',
    COUNTRY      Varchar (3) UNICODE    NOT NULL    DEFAULT "",
    REGION       Varchar (50) UNICODE,
    CITY         Varchar (50) UNICODE    NOT NULL    DEFAULT "",
    POSTALCODE   Varchar (10) UNICODE    NOT NULL    DEFAULT '0',
    ADDRESS      Varchar (100) UNICODE    NOT NULL    DEFAULT "",
    PRIMARY KEY (LOCATIONID)
)
```

```
//

CREATE TABLE BC_CR_RESERVATION
(
    ID              Varchar (10) UNICODE    NOT NULL    DEFAULT '0',
    RESERVATIONDATE  Timestamp    NOT NULL,
    DATEFROM        Timestamp,
    DATETO          Timestamp,
    STATUS          Varchar (20) UNICODE    NOT NULL    DEFAULT 'incomplete',
    EXPIREDDATE     Timestamp    NOT NULL,
    CUSTOMERID      Varchar (10) UNICODE,
    VEHICLETYPEID    Varchar (10) UNICODE,
    VEHICLEID       Varchar (20) UNICODE,
    PICKUPLOCATION   Varchar (10) UNICODE,
    DROPOFFLOCATION  Varchar (10) UNICODE,
    PRIMARY KEY (ID)
)

//

CREATE TABLE BC_CR_VEHICLE
(
    VEHICLEID    Varchar (20) UNICODE    NOT NULL    DEFAULT '0',
    MAKE         Varchar (50) UNICODE    NOT NULL    DEFAULT "",
    MODEL        Varchar (20) UNICODE    NOT NULL    DEFAULT ' ',
    FUEL         Varchar (20) UNICODE    NOT NULL    DEFAULT 'patrol',
    CONSUMPTION   Fixed (2,1)    NOT NULL    DEFAULT  0.0,
    SEATS        Smallint    NOT NULL    DEFAULT      0,
    TANKVOLUME    Smallint,
    MILEAGE      Integer    NOT NULL    DEFAULT          0,
    DESCRIPTION   Varchar (500) UNICODE,
    PICTURE      Varchar (50) UNICODE,
    AVAILABILITY  Varchar (20) UNICODE    NOT NULL    DEFAULT 'available',
    STATUS       Varchar (20) UNICODE    NOT NULL    DEFAULT '0',
    LOCATIONID    Varchar (10) UNICODE,
    TYPEID       Varchar (10) UNICODE,
    PRIMARY KEY (VEHICLEID)
)
```

```
//

CREATE TABLE BC_CR_VEHICLETYPE
(
    VEHICLETYPEID  Varchar (10) UNICODE    NOT NULL    DEFAULT '0',
    NAME           Varchar (50) UNICODE    NOT NULL    DEFAULT "",
    DESCRIPTION    Varchar (500) UNICODE,
    PRICE          Fixed (16,2)    NOT NULL    DEFAULT             0.00,
    CURRENCY       Varchar (5) UNICODE    NOT NULL    DEFAULT "",
    COUNTRY        Varchar (3) UNICODE    NOT NULL    DEFAULT "",
    PICTURE        Varchar (50) UNICODE    NOT NULL    DEFAULT "",
    PRIMARY KEY (VEHICLETYPEID)
)

//

\disconnect
```

Overview of DataSources

Definition

A DataSource is defined using a DataSource XML descriptor that contains essential information, such as the name of the underlying JDBC driver, database URL, connection pooling properties, and so on.

The DataSource XML adheres to the Document Type Definition (DTD) contained in the data-sources.dtd file. The DTD is available under *<install_directory>\<SID>\JC<instance_number>\j2ee\cluster\server0\dtd* directory of your SAP NetWeaver Application Server, Java(TM) EE 5 Edition.

For a detailed description of the *data-sources.dtd* elements, refer to the *data-sources.dtd* at **`help.sap.com`**.

In the context of the SAP NetWeaver Application Server, Java(TM) EE 5 Edition, you should use Native SQL or Vendor SQL as the SQL engine types in your DataSource definition. Also note that Vendor SQL can be used with any database that provides a JDBC-compliant driver. Native SQL is a wrapper for the vendor-specific JDBC driver and provides important enhancements such as SQL trace and statement pooling. However, it can be used only with databases supported by SAP: Oracle 9.2 and 10.2, MSSQL 8, and MaxDB 7.x.

Use

The purpose of this guide is to outline the steps you need to perform to deploy a custom DataSource, which your applications can use to connect to a database.

The SAP NetWeaver Application Server, Java(TM) EE 5 Edition comes with a default native DataSource, `SAPDEMO_DS`, which you can use in your applications.

Data Source Types

Depending on the way `DataSources` obtain connections from the underlying database driver, as well as the way they handle transactions, the following two types are distinguished:

- Driver-based

  This type of DataSource obtains Connection objects by calling the `java.sql.Driver.connect()` method implemented by the underlying JDBC driver. It returns a `Connection`, which can participate as a resource in local transactions. Besides using the `UserTransaction` object, transactions in this case can be started by invoking `setAutoCommit(false)` and completed by invoking `commit()` or `rollback()` methods on the `Connection` object.

- XADataSource-based

  This type of DataSource supports distributed transactions and is useful if multiple `Connection` objects are participating in the same transaction. The `Connection` is obtained from the vendor's implementation of `javax.sql.XADataSource`. Explicit invocations of `setAutoCommit()`, `commit()`, or `rollback()` on the `Connection` object are prohibited.

See also:

DataSource Example [Page 15]

Deploying DataSources

Use

There are two ways to deploy a DataSource:

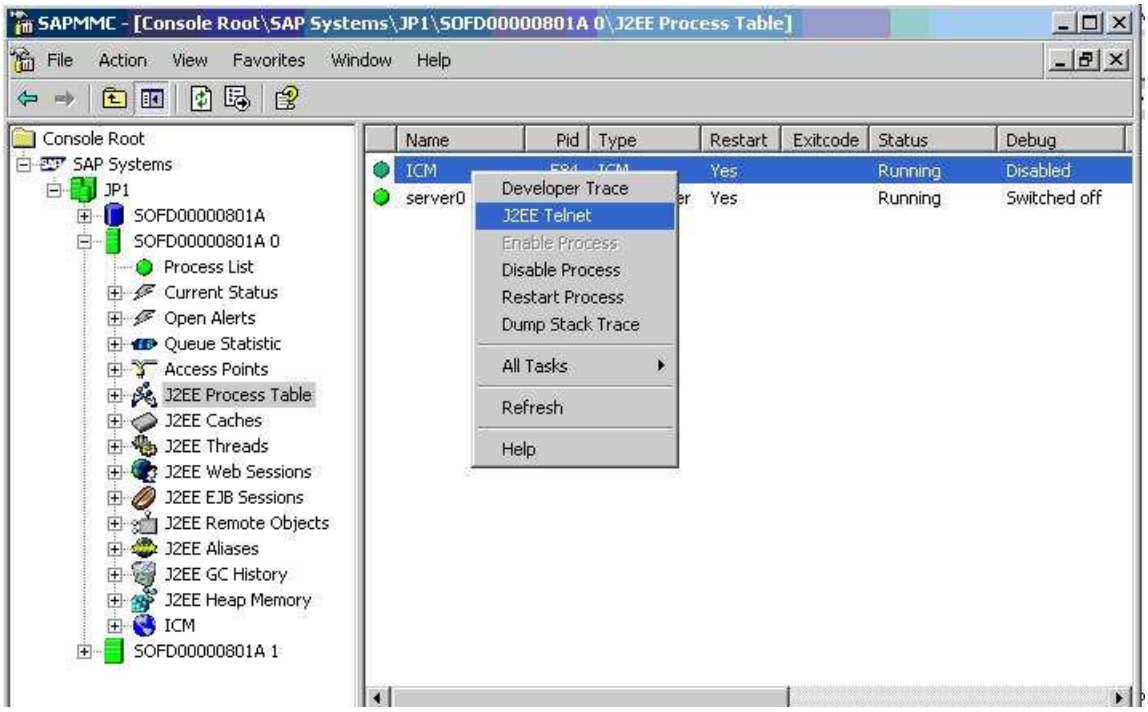
- As a standalone DataSource using the Telnet Administrator tool
- As part of a Java EE 5 application using the SAP NetWeaver Developer Studio

Deploying a Standalone DataSource

The standalone DataSource can be used by multiple applications. Its life cycle is independent from the life cycle of the applications that are using it.

To deploy a standalone DataSource, proceed as follows:

1. Open the Telnet Administrator console:
   a. Go to the SAP Management Console and open the *J2EE Process Table* for your SAP NetWeaver Application Server, Java(TM) EE 5 Edition.
   b. Choose *J2EE Telnet* from the context menu of the ICM process.



2. Login to the server by providing a valid Administrator user and password in the *User name* and *Password fields* in the Telnet Administrator console.
3. Enter `add dbpool` in the command line to add the DBPOOL shell command group to the environment.
4. Enter `make_data_source <xml_descriptor>` in the command line to deploy the DataSource defined by the `<xml_descriptor>` parameter.

   The `<xml_descriptor>` parameter specifies the path to your *data-sources.xml* file on the file system.

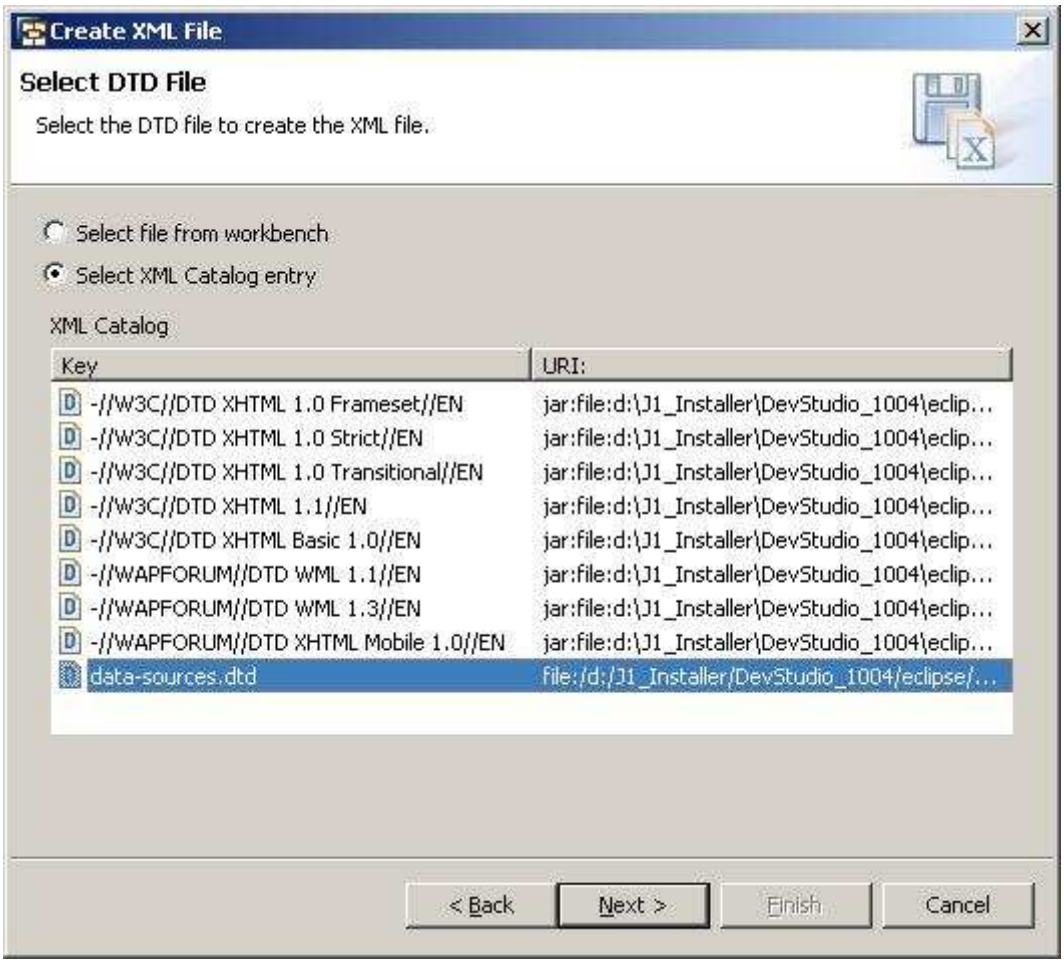
Deploying a DataSource within a Java EE 5 Application

If you deploy a DataSource as part of an application, the DataSource's life cycle will depend on the application's life cycle - that is, the DataSource will be available only if the application that deployed it is started. It can also be used by other applications by declaring resource references to it in their deployment descriptors.

To deploy a DataSource with an application, you need to add a *data-sources.xml* to the *META-INF* folder of the application's EAR file. The following procedure describes how to do this using the SAP NetWeaver Developer Studio.

To deploy a DataSource with a Java EE 5 application, proceed as follows:

1. Select the *META-INF* directory of your *Enterprise Application Project 1.5* and choose *New* → *Other* from the context menu.
2. In the *New* screen that appears, choose *XML* → *XML* and then choose *Next*.
3. In the *Create XML* file screen, select *Create XML file from a DTD file* and choose *Next*.
4. Enter *data-sources.xml* in the *File name* field of the *XML File Name* screen and choose *Next*.
5. In the *Select DTD File* screen, choose the *Select XML Catalog* entry option and choose from the *XML Catalog* list.



6. Choose *Next* and then *Finish* to create the *data-sources.xml* file.
7. Fill in the elements you need to define your DataSource. For more information, see *data-sources.dtd* at **`help.sap.com`**.

DataSource Example

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data-sources SYSTEM 'data-sources.dtd'>
<data-sources>
  <data-source>
    <data-source-name>SAPDB</data-source-name>
    <driver-name>SYSTEM_DRIVER</driver-name>
    <init-connections>1</init-connections>
    <max-connections>10</max-connections>
    <max-time-to-wait-connection>60</max-time-to-wait-connection>
    <expiration-control>
        <connection-lifetime>600</connection-lifetime>
        <run-cleanup-thread>60</run-cleanup-thread>
    </expiration-control>
    <sql-engine>native_sql</sql-engine>
    <jdbc-1.x>
        <driver-class-name>com.sap.dbtech.jdbc.DriverSapDB</driver-class-name>
        <url>jdbc:sapdb://localhost/JP1</url>
        <user-name>sapdemo</user-name>
        <password>sapdemo</password>
    </jdbc-1.x>
  </data-source>
</data-sources>
```

Deploying JDBC Drivers

Use

If you want to use a DataSource that connects your applications to a database other than the MaxDB database provided with the SAP NetWeaver Application Server, Java(TM) EE 5 Edition, you need to deploy the relevant JDBC driver for that database.

Procedure

In the Telnet Administrator console:

1. Open the Telnet Administrator and login to the server. For more information, see *Deploying a Standalone DataSource* in Deploying DataSources [Page 13].
2. Enter `add dbpool` in the command line to add the DBPOOL shell command group to the environment.
3. Enter `deploy_jdbc_driver <driver-name> <driver_archive>` in the command line, where:
   - `<driver-name>` is the name, under which this driver is registered in the system. You will use this name as a value of the *<driver-name>* tag of the *data-sources.xml* file when you define your custom DataSource.
   - `<driver-archive>` is the directory path to the archive file on the file system that contains the classes of your JDBC driver. Note that this command can have multiple `<archive-name>` arguments separated with semicolons (`;`). This is necessary when deploying drivers that consist of several JAR files.

Deployment of JMS Resources

Using the JMS Provider, you can create, remove or update deployable resources. You can define a JMS resources deployment descriptor (jms-resources.xml) and package it with your application. For more information about the schema, see *jms-resources.xsd* at **`help.sap.com`**. The XML also provides options for creating non-SAP resource references.

Here is the description of deployment of SAP JMS Resources:

- Deploy application of JMS Resources

  When an application containing a JMS resource descriptor is deployed, the respective JMS resources are automatically created if they do not already exist. If they do already exist (for example they could have been deployed by another application or they could have been administratively created), they are just associated with the application. The JMS resources can be deployed either as standalone (becoming global, that is, available to all applications), or as part of the resources of an application (becoming local for this application). Local JMS resources are available only to the application that deploys them. It is also possible to share a JMS resource between two or more applications: all applications define the respective resource with one and the same name, the resource is created during the deployment of the first application and then associated with the rest of the applications upon their deployment. Such resources will be called “shared” in this document.

- Update application with JMS Resources

  When an application containing a JMS resource descriptor is redeployed, the set of JMS resources defined in the descriptor is matched against the set of existing JMS resources. In case a resource does not exist, it is created; in case it exists, its properties are updated with the properties supplied in the descriptor (if any) and finally, in case there are resources that exist for this application but are no longer present in the descriptor, they are removed (unless these resources are global or shared). The last deployed resource can change the properties. We recommend that you make sure that you are the only one that uses these resources; to do that you have to make them local. In this way, there will be no collisions with the global resources. The redeployment of JMS resources is also called update of JMS resources.

- Remove application with JMS Resources

  When an application containing a JMS resource descriptor is undeployed, the respective JMS resources are automatically removed, unless they are global or shared.

You can use the following properties in the JMS resource descriptors:

Properties of the JMS resources

Property Name	Value Type	Relevant for	Description
agentThreadCount	int	Virtual Provider	The maximum number of threads that can work simultaneously to deliver messages to the destinations.
cleanUpServiceSleepInterval	long	Virtual Provider	The time interval in milliseconds after which the Clean Up service is called.
clientConsumerBuffer	int	Virtual Provider	The size of the buffer the client can

			use.
clientMemorySize	int	Virtual Provider	The total amount of memory the client can distribute in the buffer.
countLimitInMasterQueue	int	Virtual Provider	The limit of the number of messages in the master queue.
fetchSizeOnStartupInMasterQueue	int	Virtual Provider	The number of messages that are loaded at startup of a destination.
maxFetchSizeInMasterQueue	int	Virtual Provider	The maximum size of the loaded messages on destination startup.
sizeLimitInMasterQueue	int	Virtual Provider	The total size of the messages in the master queue.
agentKeepAliveTimeSeconds	int	Destination	The time (in seconds) to keep the agent alive after the last consumer or producer is closed.
jmsxDeliveryCountEnabled	boolean	Destination	Switches the optional message property JMSXDeliveryCount on or off.
deliveryAttemptsLimited	boolean	Queue	A flag indication if the number of delivery attempts is limited, that is if the undeliverable messages have to be moved to a dead message queue.
maxDeliveryAttempts	int	Queue	Defines the maximum number of message delivery attempts.
loadBalanceBehavior	byte	Queue	Defines the load balance behavior of the queue in case the destination has more than one consumer. The possible values are 1 (Exclusive - the registering of a second consumer will fail) and 3 (Round-robin - messages will be

			distributed among all registered consumers in a round-robin fashion).
memoryQueueMaxRowsOnStartup	int	Topic	The maximum number of messages to initially select from the database into the memory queue.
noLocal	boolean	Connection	Indicates whether the durable subscriber is local or not. If set to true, it inhibits the delivery of messages published by the subscriber's own connection.
destinationName	String	Destination	The name of the destination.