10 Chapter

Accessing Remote Data

One of MapXtreme Java's strengths is its ability to access data from remote sources for map rendering and analysis. This chapter discusses connection pooling, an efficient way to retrieve remote data.

- JDBC Connection Pooling
- Connection Pooling with MapXtremeServlet
- Connection Pooling with MapJ
- Configuring a JDB Connection
- Accessing Pooled Connections
- Security Benefits
- Connections Manager

JDBC Connection Pooling

Establishing JDBC connections to remote databases can be time consuming and resource intensive. The standard way to minimize these costs is to use connection pooling. In a connection pooling scheme a group of database connections are created and then reused and shared among many users. Since connection pooling is much more efficient than having to create separate connections for each client request, using connection pooling within MapXtreme Java is strongly recommended.

How Connection Pooling Works in MapXtreme

Typically connection pooling will be used server side with MapXtremeServlet. Additionally, MapJ can be configured to use connection pooling. Whether connection pooling is used client side or server side the behavior will be identical. When a JDBC connection is needed to access data from an RDBMS, an attempt will be made to retrieve a connection from the pool.

If the pool has an available connection, the connection will be provided to the application. Otherwise, a new connection will be created. When the task has finished using the connection it will be returned to the pool, except in the case where the pool has been filled to its maximum size. In this case, the connection will be closed, and its resources released.

Connection Pooling with MapXtremeServlet

MapXtremeServlet will utilize connection pooling when a **miconnections.properties** file is found on its classpath (the classpath of its context within its parent servlet container). Any connections described within the miconnections.properties file will be pre-started during MapXtremeServlet's init method. This guarantees that the connections will be ready and available to the first clients that visit MapXtremeServlet.

Connection Pooling with MapJ

Pooling also occurs automatically on the client tier if a miconnections.properties file is located on the classpath of the application using MapJ. Only one connection pool will be created per application and this will be utilized by all MapJ instances created within the application. A MapJ client may need to access a remote data source in the

following circumstances: 1) to perform a search method on a Layer doing local data access, 2) to obtain metadata information on a Layer doing local data accces, and 3) to do local rendering. If a connection pool is in place, it is used for each of these tasks.

MapJ also provides access to an MIConnectionPoolManager object. This can be used to establish additional connection pools at runtime.

Configuring a JDBC Connection

MapXtreme Java manages connection pooling through the miconnections.properties file. Multiple JDBC connection pools can be set up within this file. Three pooling properties can be configured when setting up each connection: 1) the number of connections to pre-start, 2) the maximum number of connections the pool can hold, and 3) the length of time a connection may remain unused before the connection is closed and its resources are returned to the application.

The following lines represent a sample entry that could be found in the miconnections.properties file.

```
Connection1_name=ProjectMaps
Connection1_driver=oracle.jdbc.driver.OracleDriver
Connection1_url=jdbc:oracle:thin:@hostmachine:port:sid
Connection1_user=mapxtreme
Connection1_password=secret
Connection1_is_xy=false
Connection1_prestart=4
Connection1_max=15
Connection1_timeout=300
Connection1_prefetch=75
```

The first line specifies the name of the connection, as each JDBC connection pool should be thought of as a "named resource". Clients will use this to get connections from the pool.

The next four lines specify the standard information needed to establish a JDBC connection: the JDBC driver to use, the connection URL of the database, and the user name and password for the database.

The next line is a peculiarity of MapXtreme Java. This entry informs MapXtreme Java whether the data source whose connections are being pooled contains spatial objects or X and Y columns of spatial data.

Note: MapXtreme Java cannot use one pool to access both X,Y and spatial object data; however, two connection pools can be set up to the same data source.

The next three settings are the connection pool properties for managing the pool's pre-start and maximum sizes, as well as the idle time before connections are closed.

Additional database specific settings may appear at the end of the list. For example, an Oracle8*i* connection may be set up to use a non-default, pre-fetch size.

The miconnections.properties file can be maintained through the stand-alone Connections Manager utility included with MapXtreme Java (see page 161). The file can also be modified manually.

Accessing Pooled Connections

Individual Layer objects within MapJ must be created in a certain way to make use of pooled connections. All DataProviderHelpers for JDBC layers share a common constructor type that takes the following input parameters:

- String URL
- Properties connectionProps (user, password, pre-fetch, etc.)
- String driverClassName

For a layer to take advantage of connection pooling it must use this form of the DataProviderHelper constructor and follow a special naming convention. The connection URL must be in the form:

```
jdbc:mipool:resource_name
```

If you are referencing the data source by a named resource, the other input parameters should be null. For example, to connect to an Oracle8*i* data source which is set up as the "ProjectMaps" named resource, you would use the following:

```
OraSoDataProviderHelper oraDpHelper = new
   OraSoDataProviderHelper("jdbc:mipool:ProjectMaps",
   null, null);
```

Security Benefits

A significant benefit to using connection pooling and named resources is that for three-tier deployments, JDBC connection information remains on the server side and is only known to MapXtremeServlet. Clients access the JDBC connections by named resource; sensitive information that describes the connection, such as user name and password, is not transmitted over the network.

Connections Manager

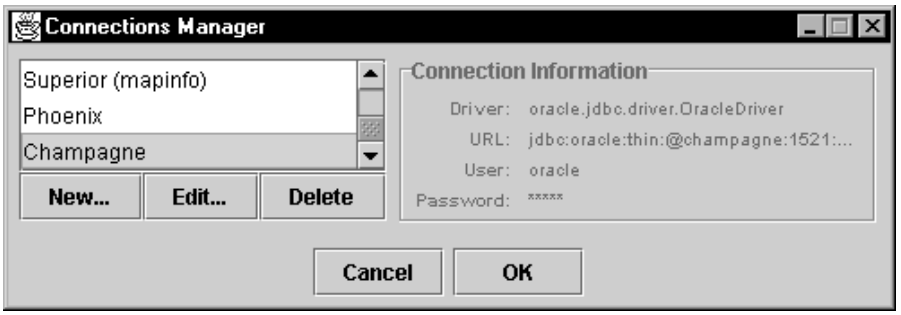
The Connections Manager is a stand-alone application that provides a user interface to manage named connections (i.e., edit the contents of the miconnections.properties file).

Run Connections Manager from the Windows Start menu or at the command line:

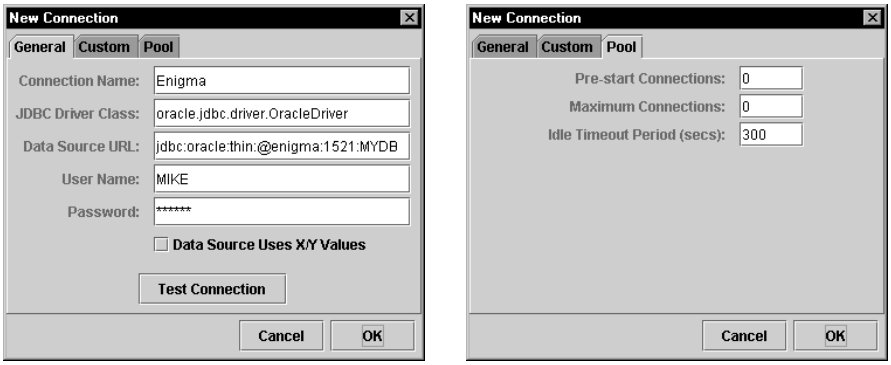
```
java com.mapinfo.dp.util.ConnectionsManager
```

In order to test your JDBC connection from the Connections Manager, be sure your JDBC drivers are in the classpath.

Connections Manager will initialize the list of named connections from the miconnections.properties file.You can create new connections or edit or remove existing connections.



The Edit dialog of Connections Manager provides three tabs for supplying information. The General tab collects the name, driver, data source URL, user and password. It also provides a Test Connection button so you can make sure the connection is good (test will fail if the appropriate driver is not in the classpath).



The Custom tab provides a place to set custom properties and values, such as pre-fetch size. The Pool tab contains the number of pre-start and maximum connections allowed, and the timeout period for idle connections.