Accessing DB2 Everyplace using J2ME devices, part 1

Skill Level: Intermediate

Naveen Balani (naveenbalani@rediffmail.com)
Developer

08 Apr 2004

This two-part tutorial assists developers in developing DB2 Everyplace mobile applications using J2ME APIs and deploying the application in the J2ME Emulator using the J2ME toolkit. Part one deals with developing the mobile application.

Section 1. Introduction

Overview

This tutorial assists developers with building DB2 Everyplace mobile applications using J2ME DB2 Everyplace MIDP ISync APIs and deploying the application in the J2ME environment using J2ME Toolkit.

Part one of this two-part tutorial uses the DB2Everyplace J2ME MIDP ISync Client APIs to create a Sample Address Book database on a J2ME Emulator/Device from a remote DB2 database. The Address Book database on the J2ME Emulator/Device is stored using J2ME Record Store Management System.

Part two builds an address book application that accesses the Address book database, performs updates to it and then synchronizes it with the remote DB2 Address database. We shall use DB2 Everyplace MIDP APIs to access the Address Book database and perform updates to it.

Prerequisites

The developers must have knowledge and an understanding of DB2 Everyplace and J2ME.

This article requires that you have already installed DB2 Everyplace Enterprise Edition 8.1.4,DB2 Everyplace Software Development Kit 8.1.4 and J2ME ToolKit.

Software requirements

The software required for the tutorials is:

- Windows 200 with Service Pack 3 or above.
- DB2 Everyplace 8.1.4 Enterprise Edition from DB2 Everyplace Enterprise Edition
- DB2 Everyplace 8.1.4 Software Development Kit from DB2 Everyplace Software Development Kit
- J2ME ToolKit 2.1 from J2ME Wireless Toolkit 2.1

Section 2. Getting started

Overview of the DB2 Everyplace solution

IBM DB2 Everyplace is part of IBM's solution for mobile computing. The DB2 Everyplace solution consists of the following components:

- **DB2 Everyplace database**: A relational database system designed for small handheld devices.
- **DB2 Everyplace Sync Server**: A bidirectional synchronization server that moves data between the DB2 Everyplace database and the existing enterprise databases.
- **DB2 Everyplace Mobile Application Builder**: A rapid application development tool for creating DB2 Everyplace applications running on handheld devices.

The first component of DB2 Everyplace is its handheld database engine. The engine is a true, relational database engine delivering persistent storage for record sets and the ability to modify and retrieve records. Data in a DB2 Everyplace database can be accessed using several different methods. One way to access data is to use the

Command Line Processor (CLP) to issue SQL statements. A second way to access data is to write your own applications using ODBC, JDBC call-level interface functions or DB2 ADO.NET providers. DB2 Everyplace uses SQL to modify and access data.

The second component of DB2 Everyplace is the Synchronization server, or Sync Server. Sync Server is a client/server program that manages the data synchronization process from the handheld DB2 Everyplace database to the source DB2 database. The source DB2 database can be any DB2 UDB server platform. Sync Server enables two-way data synchronization from the handheld database to a DB2 UDB database, as well as from the DB2 UDB database to the handheld database.

The third component of DB2 Everyplace is the Mobile Application Builder (MAB). MAB is an integrated toolkit for developing DB2 Everyplace applications running on handheld devices such as Palm, Symbian, Quartz and Windows CE.NET.

What are MIDlets?

A *MIDlet* is an application written for the Mobile Information Device Profile (MIDP), which is part of the Java runtime environment for mobile information devices such as phones and PDAs. MIDP provides the core application functionality required by mobile applications, such as the user interface, network connectivity, local data storage, and application lifecycle management. It is packaged as a standardized Java runtime environment and set of Java APIs.

The MIDP specification supports HTTP client capabilities, which allows a MIDlet to access remote services through HTTP. MIDP provides user interface APIs for display purposes, giving applications direct control over the user interface -- similar to Java Swing.

The MIDlet lifecycle

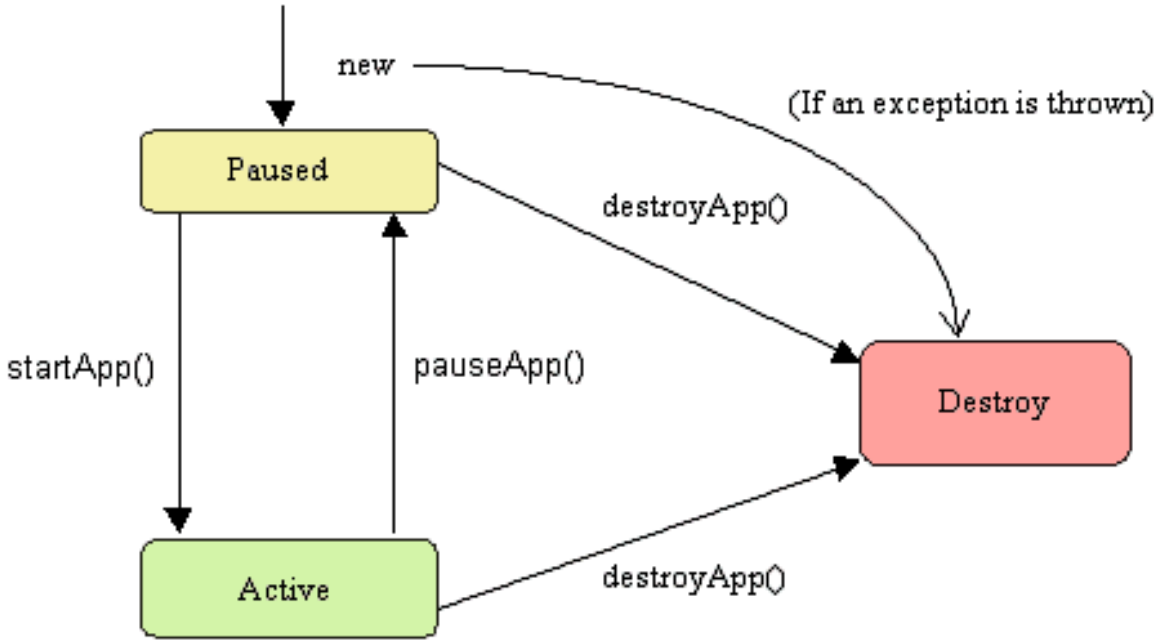
A MIDlet is managed by the Java Application Manager, which executes the MIDlet and controls its lifecycle. A MIDlet can be in one of three states: *paused, active,* or *destroyed.*

When first created and initialized, a MIDlet is in the paused state. If an exception occurs in the MIDlet's constructor, the MIDlet enters the destroyed state and is discarded.

The MIDlet enters the active state from the paused when its `startApp()` method call is completed, and the MIDlet can function normally.

The MIDlet can enter the destroyed state upon completion of the `destroyApp(boolean condition)` method. This method releases all held resources and performs any necessary cleanup. If the `condition` argument is `true`, the MIDlet will always enter the destroyed state.

The following figure illustrates the various states of the MIDlet lifecycle.



We'll see all of these processes in action in our sample application in Lifecycle methods.

J2ME record management system

The J2ME *record management system (RMS)* provides a mechanism through which MIDlets can persistently store data and retrieve it later. In a record-oriented approach, J2ME RMS comprises multiple record stores.

The record store is created in platform-dependent locations, like nonvolatile device memory, which are not directly exposed to the MIDlets. The RMS classes call into the platform-specific native code that uses the standard OS data manager functions to perform the actual database operations. Each record in a record store is an array of bytes and has a unique integer identifier.

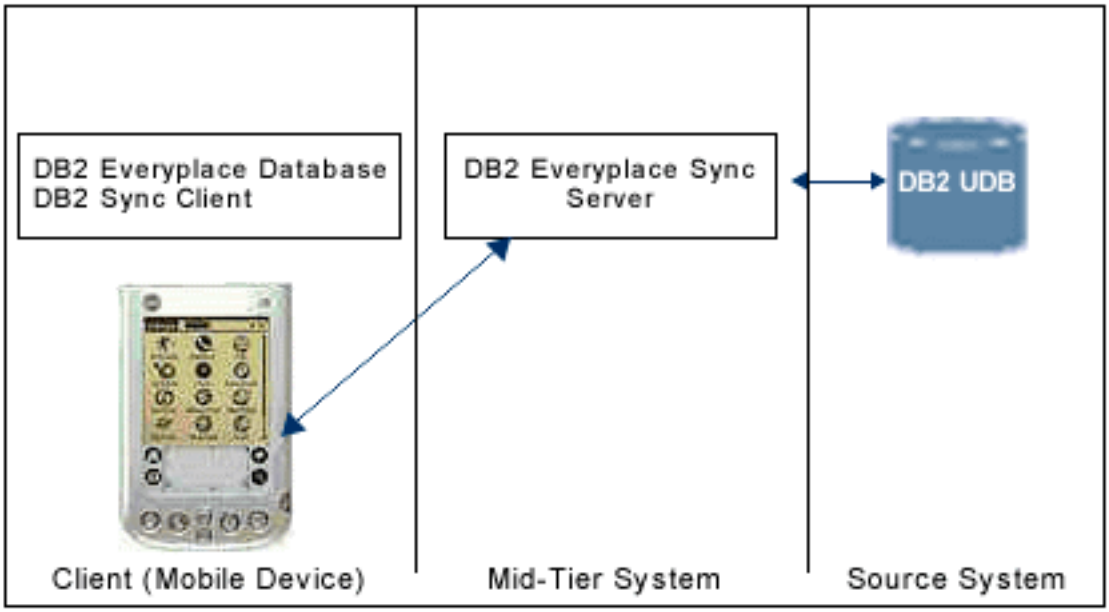
The `javax.microedition.rms.RecordStore` class represents a RMS record store. It provides various methods to manage as well to insert, update and delete records in a record store.

Overview of DB2 Everyplace J2ME MIDP Synchronization

The J2ME MIDP ISync Client allows you to build applications that synchronize subscriptions to the MIDP Record Store Management System (RMS). The J2ME MIDP ISync Client is a set of libraries that work with the DB2 Everyplace Sync Server to simplify the synchronization of relational data between enterprise databases and MIDP 2.0 enabled devices. The Sync Server manages the movement of data to and from the MIDP device.

The J2ME MIDP Synronization APIs creates the necessary Record Management System for our database application during synchronization on the J2ME MIDP Enabled Device/Emulator.

Figure 2 shows the interaction between the mobile client, the sync server, and the source database.



Section 3. Configuring the DB2 Everyplace sync solution

Configuring the DB2 Everyplace sync solution

Here is an overview of steps required to carry out the synchronization process.

1. Create the source (DB2) database and tables for synchronization on the server.
2. Create synchronization objects, using the Mobile Devices Administration Center (MDAC).
3. Set up the Mobile Device (J2ME Emulator in our case) for synchronization.

Create the source (DB2) database and tables on the server

We need to create source tables on the server. These "source" tables are the tables on the DB2 server database with which we synchronize our wireless data. We'll be using the data from the address book that we created in the previous article. To create the source tables:

1. Create the source (DB2) database and tables for synchronization on the server.
2. Create synchronization objects, using the Mobile Devices Administration Center (MDAC).
3. Set up the Mobile Device (Windows.NET Emulator in our case) for synchronization.

Create the source (DB2) database and tables on the server

We need to create source tables on the server. These "source" tables are the tables on the DB2 server database with which we synchronize our wireless data. We'll be using the data from the address book that we created in the previous article. To create the source tables: \

1. At the Windows command prompt, enter **db2cmd**.
2. Create the required database and tables, and populate it with our sample data by entering the following statements:
   db2 create database address
   db2 connect to address
   db2 CREATE TABLE ADDRESS(ADDRESS_ID Char(30) not null primary key,FirstName Char(30), LastName Char(30),StreetAddress char(50),PhoneNumber char(15))
   db2 INSERT INTO ADDRESS VALUES ('0000000001','ALEX','STEWART','1500 DEC ROAD, CAF CITY , CA

94357','510-999-7898')
db2 INSERT INTO ADDRESS VALUES
('0000000002','RAJ','MALHOTRA','161 SKY STREET CAF CITY , CA
94557','415-234-7898')
db2 disconnect address
This creates the address database and a table called address in that database.

Create synchronization objects using MDAC

A synchronization object contains information about aspects of the synchronization process. There are six types of synchronization objects that can be configured for synchronization.

1. **User**: A user who uses the DB2 Everyplace Sync Server to synchronize data between a source (the enterprise system) and a target (the mobile device). You assign a user to a group to provide access to the subscriptions that are defined in the group's subscription sets.

2. **Group**: A group of users with similar mobile data synchronization needs. You define synchronization characteristics for each group, such as which applications the users in the group need to access to perform their jobs and what subsets of enterprise data they need to access.

3. **Subscription**: A specification for what information in a source database or server is to be replicated to a target database (the DB2 Everyplace database on the mobile device). You can create two types of subscriptions:
a) File subscriptions for files stored at the source server.
b) Table subscriptions in the source database using either DataPropagator or JDBC subscriptions.
DataPropagator subscriptions provide users with access to data in source tables on a DB2 server, while JDBC subscriptions provide users with access to data in source tables on a data source with a JDBC interface, including Oracle, DB2, Microsoft SQL Server, Informix, Sybase, and Lotus Domino.

4. **subscription set**: A collection of subscriptions. To provide group members with access to the data and files defined in subscriptions, you collect the subscriptions together in a container called a subscription set, then assign this container object to the group. When users start the synchronization client software on the device, they choose which subscription set to synchronize. The menu of subscription sets that

appears on the client is created from the list of subscription sets that is associated with the user's group.

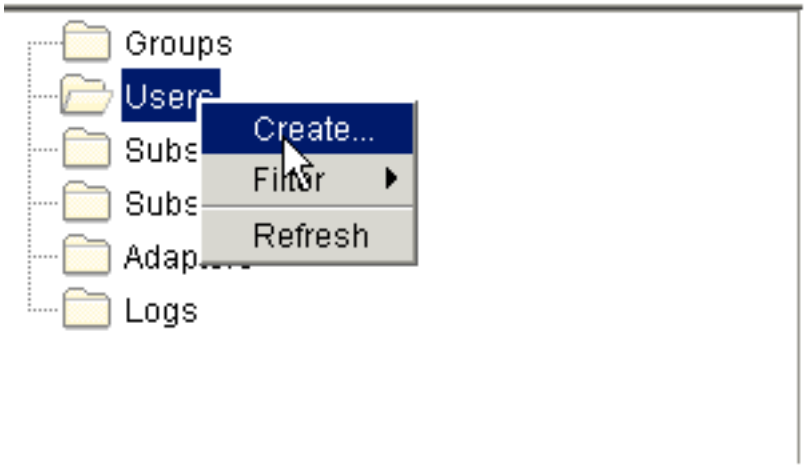
5. **Adapter**: An adapter is used to synchronize and communicate with the Sync Server. A collection of adapters is included for synchronizing files, relational data (with DB2), relational data (with JDBC), and remote query and stored procedure functionality.

6. **Log**: After you implement mobile data synchronization, you can monitor any synchronization problems using the error messages written to the error log.

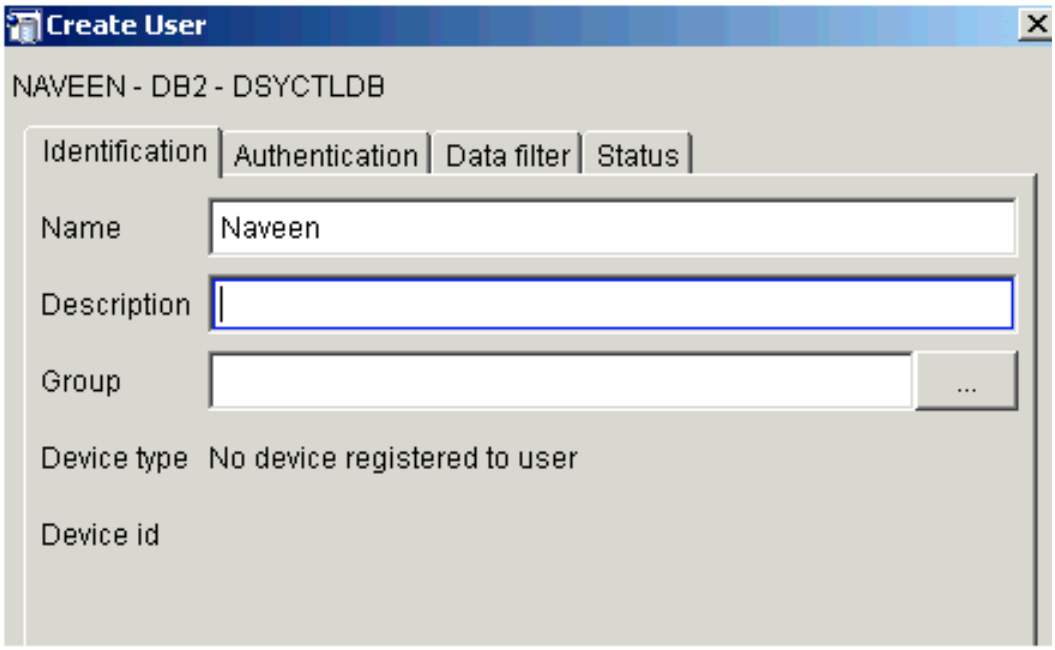
For our synchronization process, let's create a user, a group, a subscription, and a subscription set. Use the MDAC to do this. All these synchronization objects are stored in the synchronization control database, DSYSCTLDB, which was created when you installed the DB2 Everyplace Sync Server.

Creating users

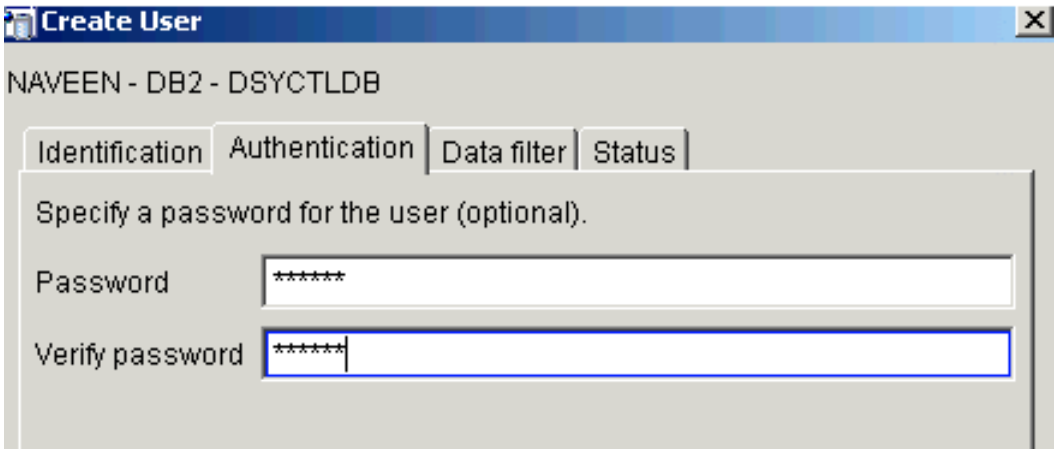
1. Click on **User -> Create**, as shown below:



2. Enter the name of the mobile user, in our case, **Naveen**, then click on the **Authentication** tab.



3. In the Authentication tab, enter a password. This password will be required as part of authentication process from the Mobile Client to the Sync Server.



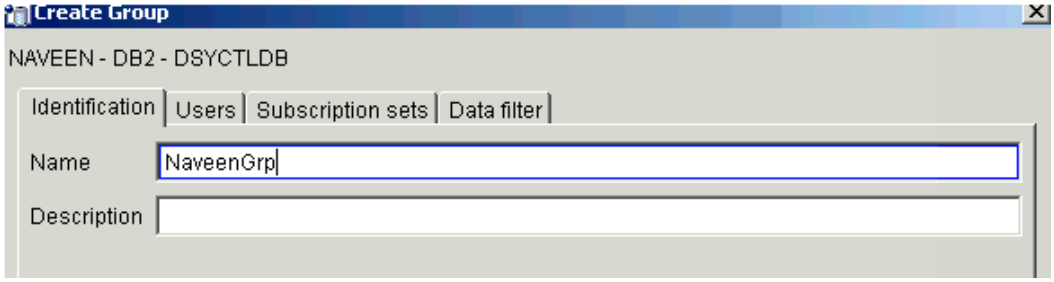
4. Click **OK** to finish creating the User object.

Next we must add this user to a user group. A group is required to provide access to the subscriptions defined in the group's subscription sets.

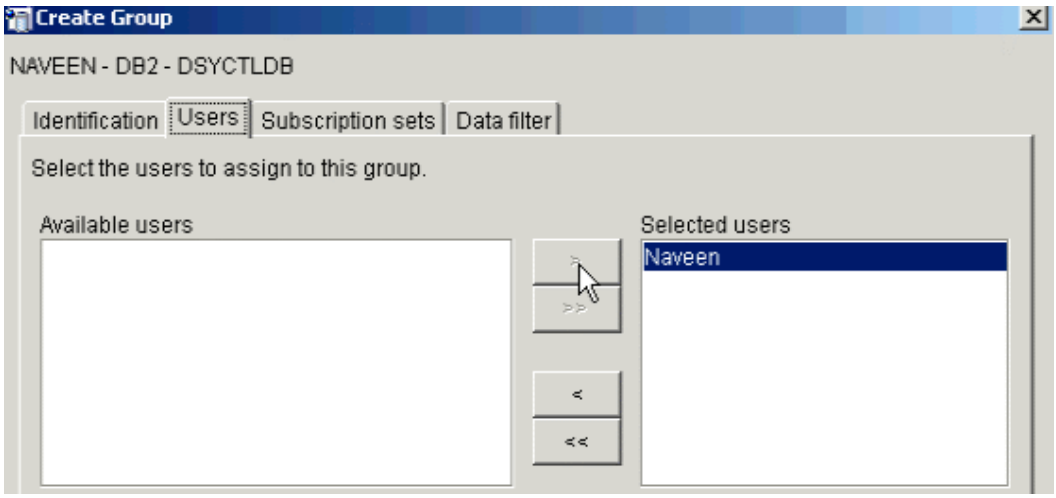
Creating groups

1. From MDAC, click on **Group -> Create**.

2. Enter the name of the group. In our example, we use **NaveenGrp**.



3. Click on the **Users** tab, then move the user Naveen from Available users to **Selected users**, then click **OK**.



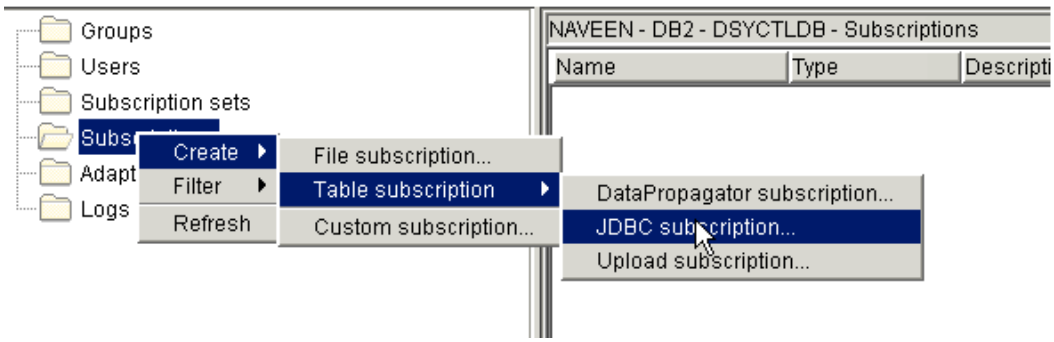
We've now added Naveen to a group. Let's now create the subscription object.

Creating a subscription

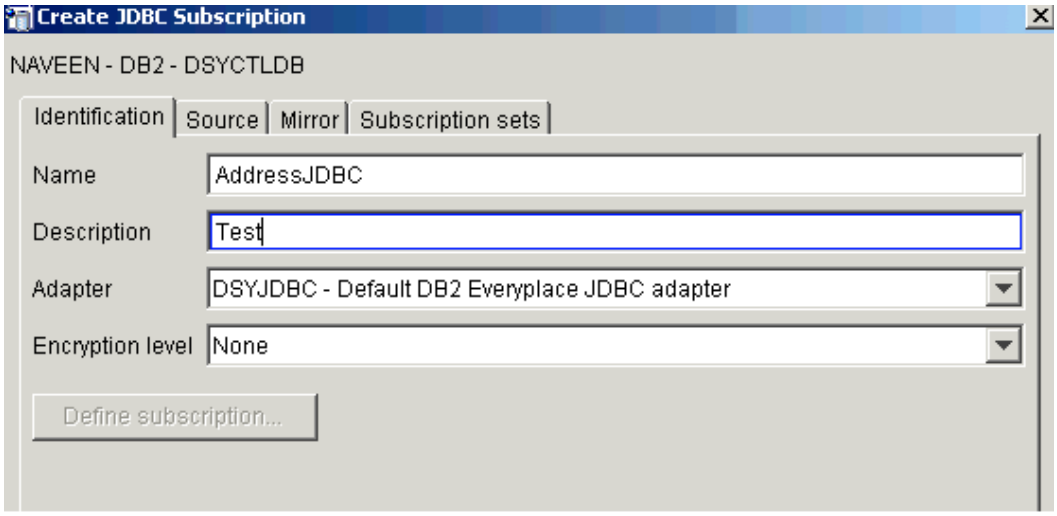
This subscription process involves the following steps:

1. Creating the JDBC subscription
2. Creating a mirror database
3. Mapping source and DB2 Everyplace table

1. Click subscription, then select **Create -> Table subscription -> JDBC**

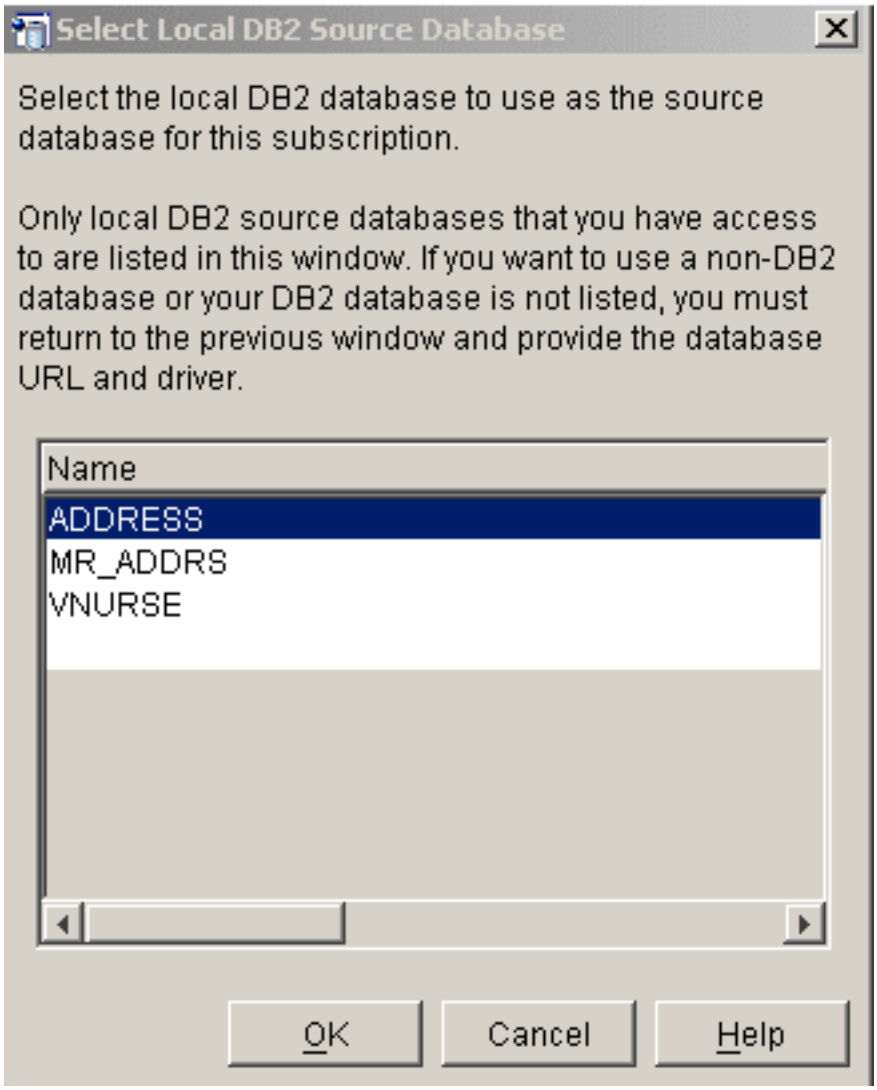


2. Enter a name for the JDBC subscription. For our example, we are calling it **AddressJDBC**. For encryption, select **None**, and leave the Adapter value as it is.



3. In the Source tab, enter a user ID and password that has access to the DB2 database. For Driver, select **IBM DB2 UDB local**, because we have installed DB2 on a local machine. (You would choose remote if DB2 UDB is located remotely.)

4. Click on the tab next to Database URL, which brings up the Select Local DB2 Source Database window, shown here. Select the **ADDRESS** database, then click **OK**.



Creating a mirror database

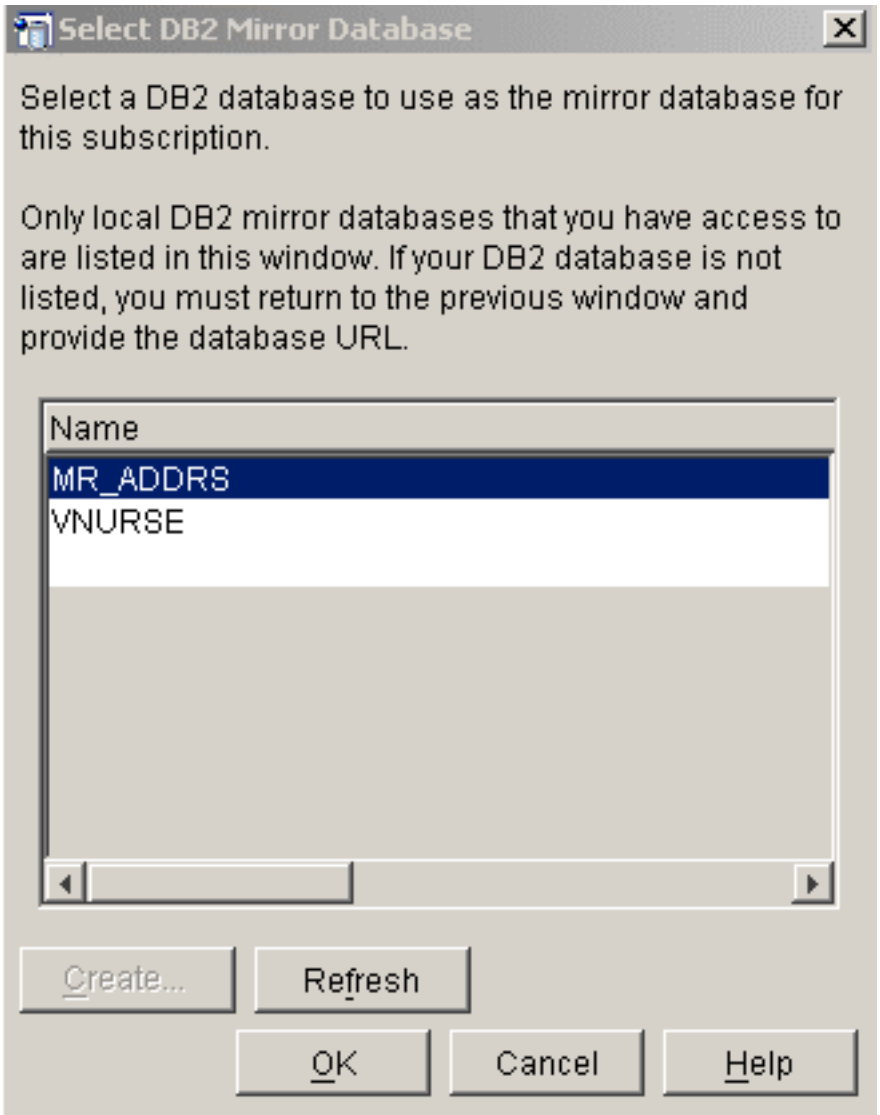
Next we create and specify a mirror database for our application. The mirror database is required for staging operations to improve the throughput capacity of synchronization requests. Changes can be staged while other updates are taking place. The mirror database can also be used to resolve any conflicts that arise because of access from multiple clients. To create the mirror database:

1. Open up the DB2 command prompt and type in the following, which creates the mirror database `ie mr_addrs`.
   db2 create database MR_ADDRS

2. Click on the **Mirror** tab on Create JDBC subscription Window, and enter a

user ID and password that has access to the DB2 database.

3. Click next to Database URL, which brings up the select mirror database window. Select **MR_ADDRS** as the mirror database as shown below and click **OK**.

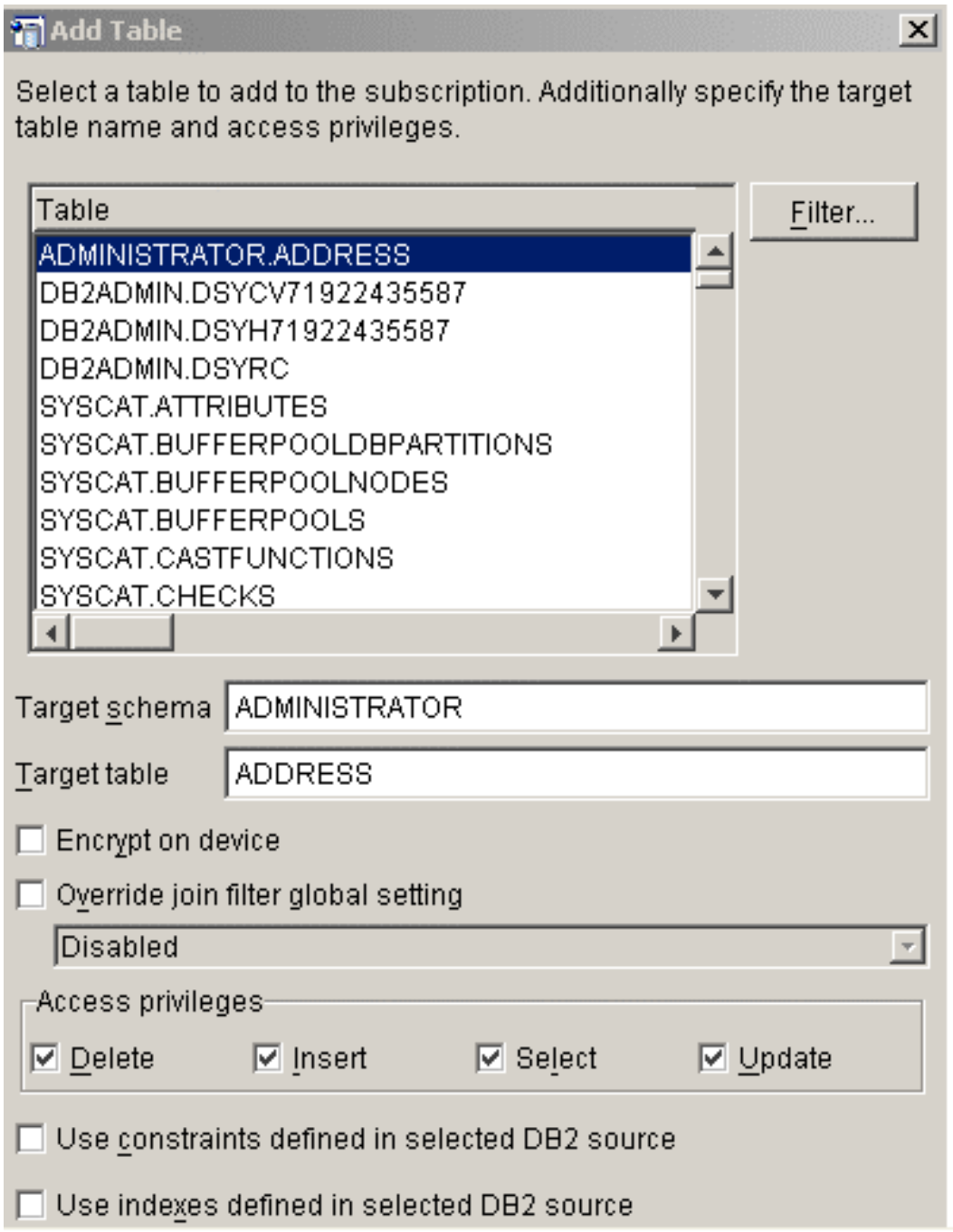


Mapping source and DB2 Everyplace table

Now we must map the source table (DB2) with the DB2 Everyplace table. To do this:

1. Click on the **Define subscription** button of the Create JDBC subscription window.

2. Click **Add**, which invokes the window shown here:

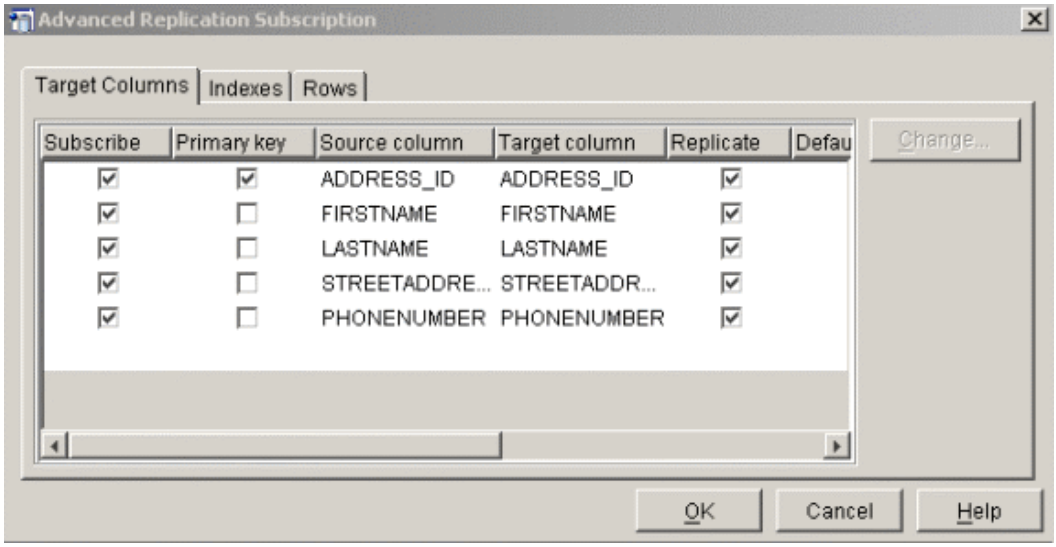


3. Select **ADMINISTRATOR.ADDRESS** as the source (DB2 UDB) table. By default, the target table is set to same as the source table. In our case, the target table maps to the DB2 Everyplace database table that is installed on the client side. Because our source tables and target table have the same names and have a one-to-one mapping it will work for us.

If you have a different target table you need to specify it in the Target table field. If your target table doesn't have a schema, the schema name of the target default will be same as the source schema.

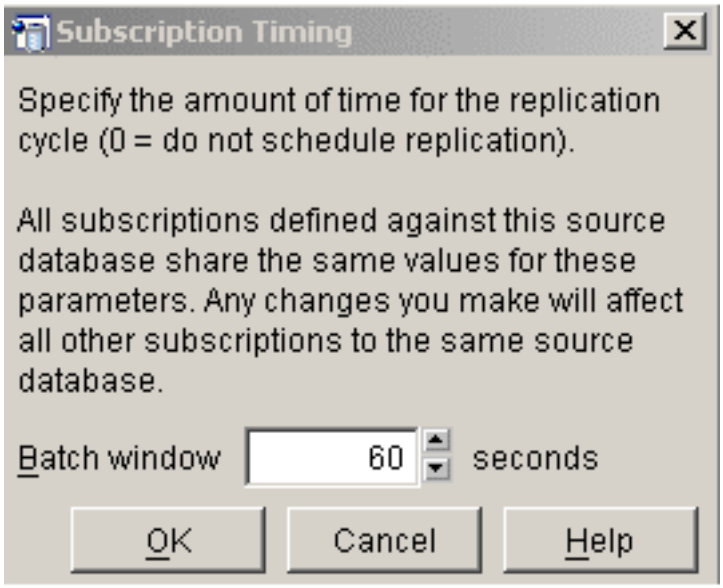
4. Click **Add** to add this information.

5. Select **ADMINISTRATOR.ADDRESS** on the Define subscription window and click **Advanced**.



This shows you the mapping of source table columns with target table columns, and it lets you indicate which fields you want to replicate. Because our column names are the same we don't need to do anything. Otherwise, we would have to select a particular field and use the Change option. Click **OK** to close the window.

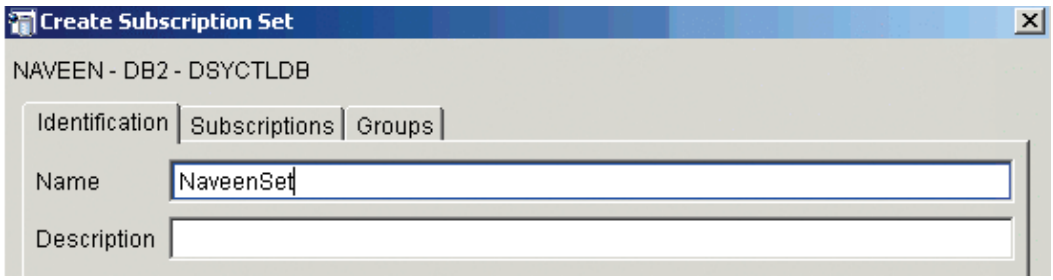
6. Select **Timing** and the subscription Time window will open. Enter a value of **60** in the batch window field. This specifies the replication timing interval.



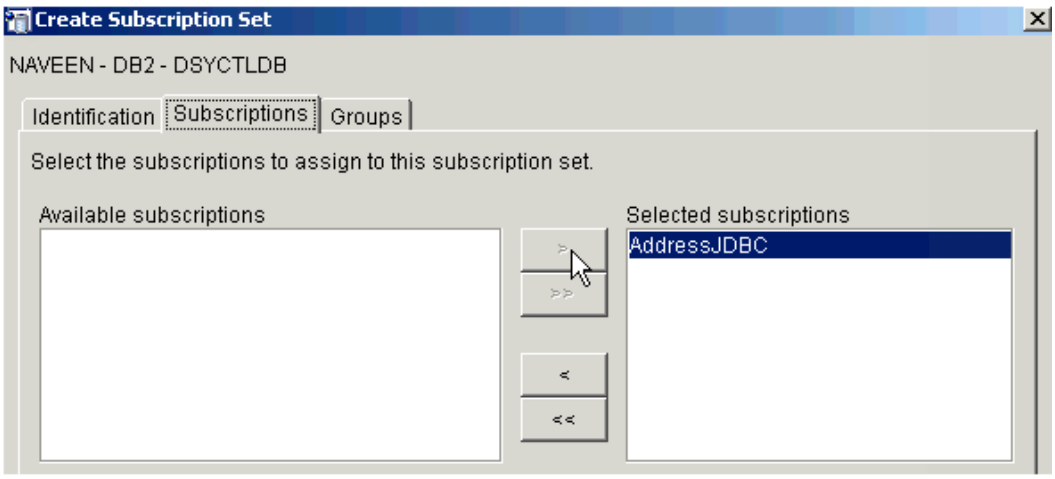
Now we have finished describing our subscription object, and it's time to group our subscriptions and users group into a subscription set.

Creating a subscription set

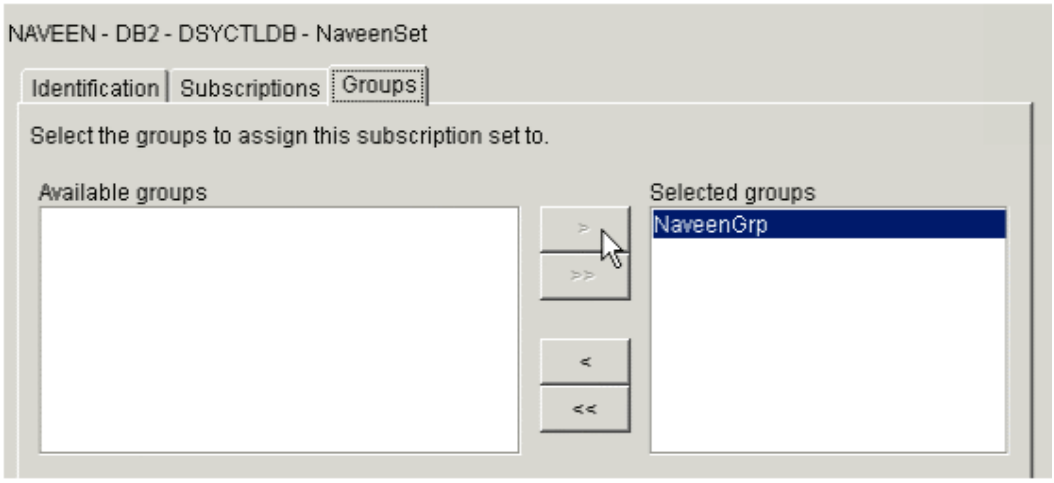
1. Click on the **Define subscription** button in the Create JDBC subscription window.
2. Click on the **subscription Sets** section in MDAC.
3. Select **Create** to create a new subscription set, which invokes the window shown below:



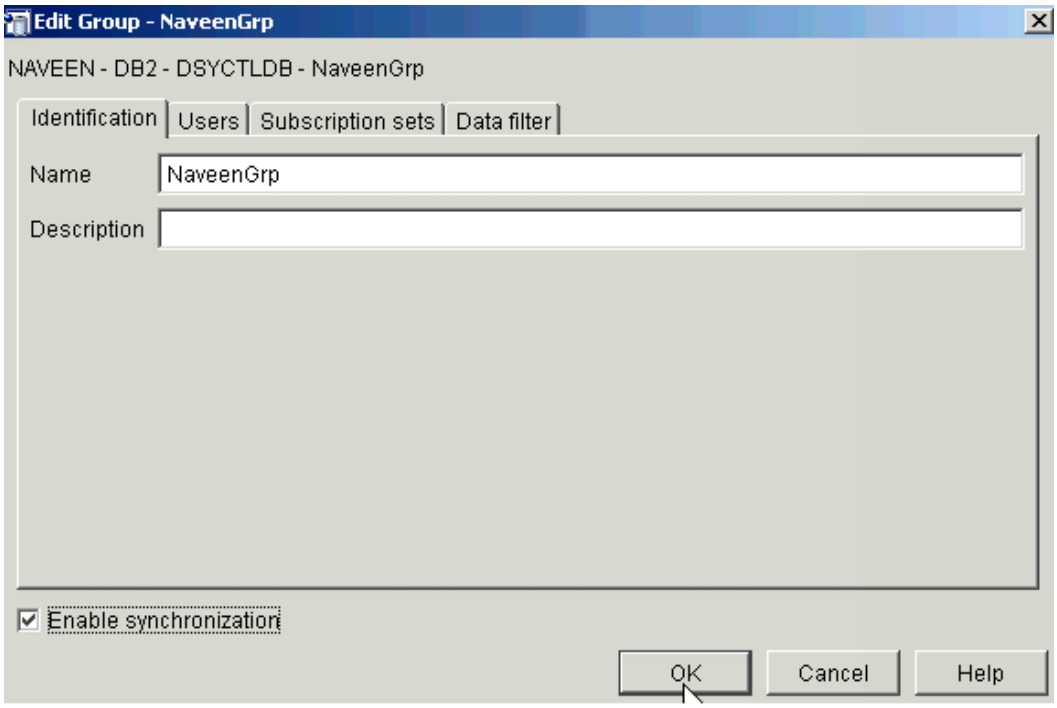
4. Enter the subscription set name in the Name field of the Identification tab.
5. Click on the **subscriptions** tab, and add **AddressJDBC** subscription to the selected subscriptions tab:



6. Similarly, click on the **Groups** tab and add the **NaveenGrp** to the Selected groups pane as shown below, then click **OK**.



7. Enable synchronization for our user groups:
   a. Choose the **Group** section.
   b. Click on **NaveenGrp**, and select Edit, which invokes the window shown



c. Select the Enable synchronization checkbox and click **OK**.

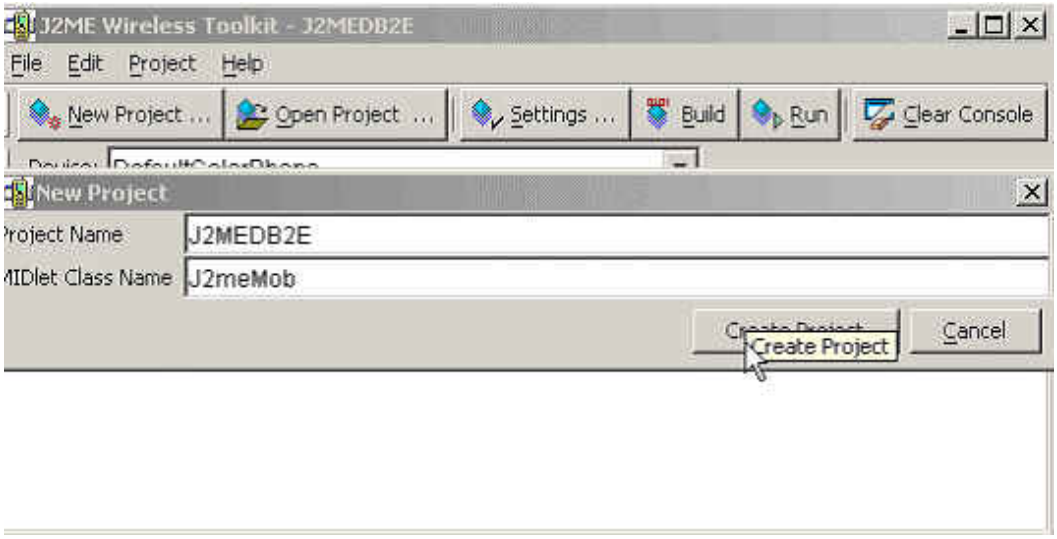
Our subscription sets are now configured. Next we move on to build our application.

Section 4. Building the address book application for sync

Creating the MIDlet

In this section we shall import our existing sample address book midlet application. Extract the Sample-J2MEDB2E1.ZIP from the resource section into any directory, say c:\. A folder J2MEDB2E will be created. First, let's create a new MIDP project in the J2ME toolkit. Follow these steps:

1. Navigate to **Start=>Programs=>J2ME Wireless Toolkit 2.1=>Ktoolbar.** The toolkit window will open.
2. Create a new project using Ktoolbar. Enter `J2MEDB2E` as the project name and `J2meDB2EMob` as the MIDlet class name. Click on **Create Project.**



3. The next window will give you the opportunity to specify the project settings. Click **OK** to accept the deafults.

4. Copy the IsyncMidp.jar file from DB2EveryplaceSDKinst\Clients\MIDP\lib to c:\wtk21\apps\ J2MEDB2E \lib. The IsyncMidp.jar file contains the necessary DB2E MIDP Synchronization classes and DB2 specific RMS classes.

5. Copy the source file, `J2meDB2EMob.java`, from `c:\J2MEDB2E` to `c:\wtk21\apps\J2MEDB2E\src`. (Remember, the J2ME Wireless Toolkit is installed in the `C:\wtk21` path.) J2meDB2EMob.java is a MIDlet application that uses the DB2 MIDP Sync APIs for synchronizing with our address database application. When synronization is done , the DB2 MIDP Sync APIs automatically persists our address book data to J2ME devices by using the available Java persistence API, this being the Record Management System (RMS) for MIDP. Thus this process creates our necessary address book record store.

6. Click the **Build** button on the Ktoolbar. You will receive the following message:

```
Project settings saved
Building "J2MEDB2E"
Build complete
```

Congratulations. You have successfully built the `J2meDB2EMob.java` MIDlet.

Section 5. Our MIDlet application: Code review

Defining synchronization and command interfaces

Let's take a look at the beginning of J2meDB2EMob.java. The line numbers I'll use in this section are only for reference in my discussion; these numbers do not match up with those in the actual source code, given that this isn't a complete listing.

```
Line 1: public class J2meDB2EMob extends MIDlet implements implements CommandListener,
Runnable {

Line 2: // Sync Client API instance

                private ISyncProvider provider;

                private ISyncService service;

                private ISyncConfigStore config;

                private ISyncDriver isync;


Line 3: private String storeName = "ADDRESS";


Line 4: private Command backButton;

                private Command exitButton;

                private Command showStartButton;

                private Command syncButton;


Line 5:  public J2meDB2EMob () {
                display                 = Display.getDisplay(this);
                backButton          = new Command("Back", Command.BACK, 2);
                syncButton          = new Command("Sync", Command.SCREEN, 1);
                showStartButton          = new Command("J2meDB2EMob", Command.SCREEN, 1);
                exitButton = new Command("Exit", Command.SCREEN, 2);
        }
```

In the listing, Line 1 defines the `J2meDB2EMob` class, which extends the `MIDlet` class and implements `CommandListener` for capturing events.J2meDB2EMob class also implements `Runnable` interface to carryout synchronization process in another thread.

Lines 2 define Synchronization Interfaces, Line 3 defines the database that needs to be remotely synchronized, Line 4 and 6 defines control elements for capturing user input.

Executing the commands

```
Line 6: public void commandAction(Command c, Displayable s) {
Line 7: if (c == syncButton)
                                    {
                                                    performSyncronization();
                                                    return;
                                    }

Line 8: private void performSyncronization()
                    {

                                                    syncThread = new Thread(this);
                                                    syncThread.start();
                                                    return;
                    }
```

Line 8 performs a check if the user has clicked the synchronization button, if yes the performSyncronization method is called. The performSyncronization method at Line 9 starts the synchronization thread which carries out the synchronization process.

Lifecycle methods

Lines 9 through 11 provide the MIDlet lifecycle methods that we discussed in The MIDlet lifecycle. Every MIDlet class needs to provide these methods.

```
Line 9 : public void startApp () {
                    Hashtable ht = new Hashtable();
                    ht.put("isync.user", "Naveen");
                    ht.put("isync.password", "Naveen");

                    String lang = getAppProperty("Db2eSyncLang");
                    if (lang != null && lang.equals("jp"))
                    ht.put("isync.encoding", "UnicodeBig");
                    ht.put("isync.trace", ISync.TRACE_ON);

                    service = provider.createSyncService("http://localhost:8080", ht);

          }
Line 10: public void pauseApp () { }
```

```
Line 11: public void destroyApp (boolean unconditional) {
 }
```

Line 9 the StartApp() method initializes Synchronization service interfaces. The parameters passed to provider.createSyncService() includes the DB2 synronizaton server ipaddress along with the port number and a hashtable consisting of the user and password information authorized to carry out the synchronization.

Synchronization process method

```
Line 12: public void run(){


                                                    int rc = isync.sync();
                                                    switch (rc)
                                                    {
                                                    case ISync.RTN_SUCCEEDED:
System.out.println("Synchronization succeeded ");
                                                          break;

                                                      case ISync.RTN_CANCELED:
System.out.println("Synchronization canceled ");
                                                                    break;
                                                    }

Line 13: readRecords(storeName);

 }
```

The run method at Line 12 carries out the synchronization process. The run method is called when the synchronization thread is started at line Number 9. Line 13 calls the our method readRecords (storeName) which executes through the address record store created during the synchronization process and prints outs the each address book records.

Now that we've seen how the code works, let's see it in action as we run the application on an emulator.

Section 6. Running the address book application

Running the application

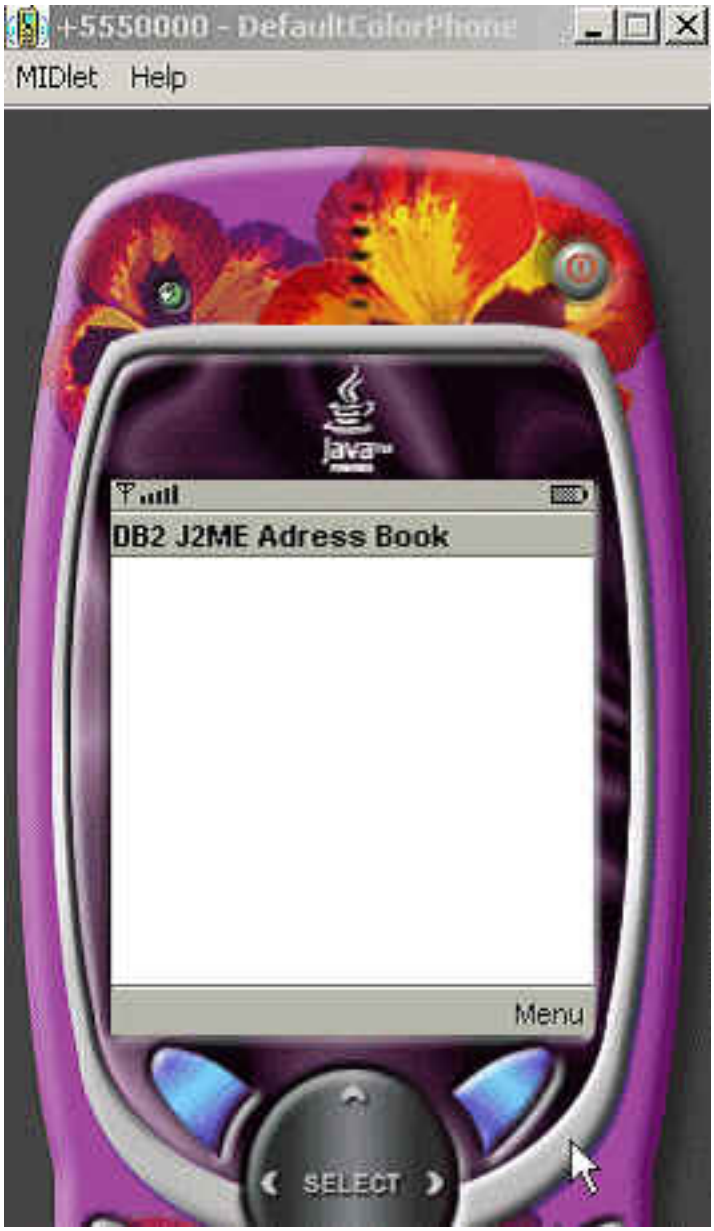
Before running the application, start the Sync Server by navigating to **Program Files**

-> IBMDB2Everyplace -> Start Servlet for SyncServer.

To run the sample J2MEDB2E application, click the **Run** button on the Ktoolbar. The default emulator shown in the following figure should appear.

Running the address book application

On launching the application, you should see the following screen:

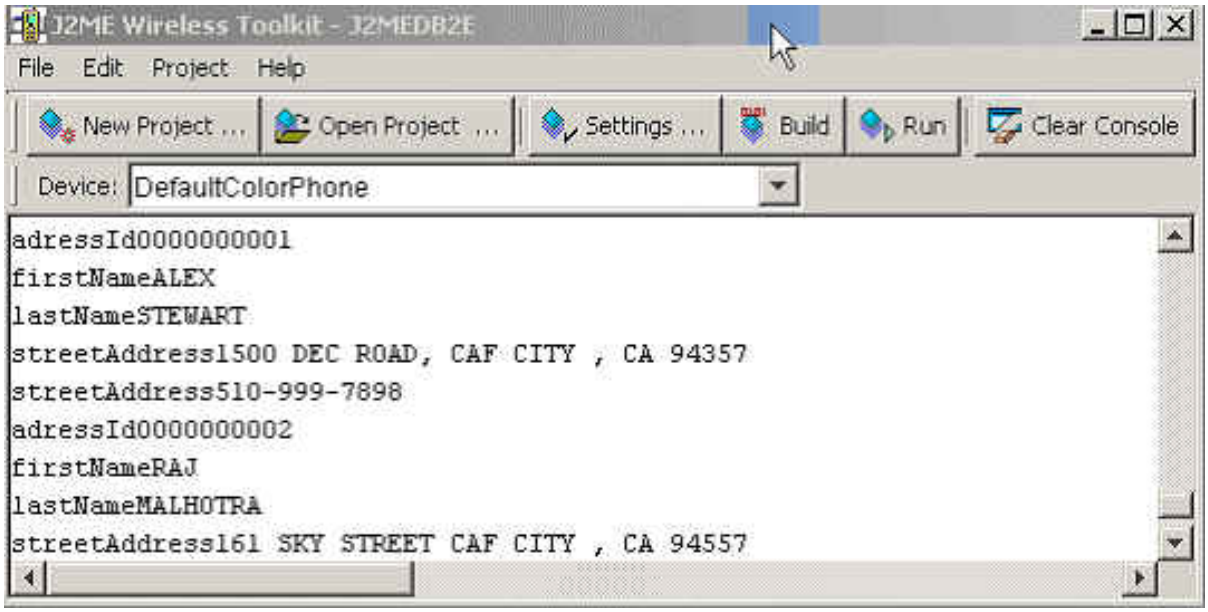


Click on the Menu. The following drop-down will appear; choose the Sync option to perform synchronization.

Building and running the application

On clicking the Sync button, the Synronization process will start and J2ME Emulator will try to connect to DB2 Synchronization server. You will receive a warning message during connecting as shown below. Click **OK** on the screen to proceed with a network connection to the DB2 Synchronization server to carry out the synchronization process.

When synronization is complete, the DB2 MIDP Sync APIs automatically persists our address book data to J2ME devices by using the available Java persistence API, this being the Record Management System (RMS) for MIDP. Once synronization is complete, our readRecords method is called, which reads all our address book records created on the J2ME Emulator/Device and displays it to the output console. At the Ktoolbar console we can view all the records read. Given below is the output of the Ktoolbar console.



Thus we have successfully created our address book record store. In the next part of the tutorial we will build an address book application that accesses this local address book record store, performs updates to it and than synchronizes it with the DB2 database.

Section 7. Summary

Wrap up

In this tutorial we have successfully developed and deployed our address book record store on J2ME emulator using the J2ME ToolKit. Similarly you can create record stores on J2ME devices using the synchronization process, so remote data is available on our local J2ME Device. In part two of the tutorial, we will access our address book record store, perform updates to it using DB2 JM2E MIDP APIs and later synchronize the updates with our remote DB2 address database.

About the author

Naveen Balani
Naveen Balani spends most of his time designing and developing J2EE Based

products. He has written various articles for IBM in the past covering topics from JMS, SOA, SOAP, Web services, Architecture Patterns, MQSeries, WebSphere Studio, XML, JAVA Wireless Devices and DB2 Everyplace for Palm, Java-Nokia and Wireless Data Synchronization.