Mobile and Wireless Systems Programming

RMS
Record Management System

Table of contents

- MSA
  - Record Stores (MIDP)
  - FileConnection API (JSR 75)
  - Personal Information Management API : Contacts, Calendar, To-Do list (JSR 75)
- Record stores
  - Like a database table and consists of records, which are simply byte arrays

javax.microedition.rms

- Interface :
  - RecordComparator
  - RecordEnumeration
  - RecordFilter
  - RecordListener
- Class :
  - RecordStore
- Exception :
  - InvalidRecordIDException
  - RecordStoreException
  - RecordStoreFullException
  - RecordStoreNotFoundException
  - RecordStoreNotOpenException

static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary)

- Open (and possibly create) a record store associated with the given MIDlet suite
- Since MIDP 1.0

```
RecordStore rs;
try {
        rs = RecordStore.openRecordStore("cache", true);
} catch (RecordStoreException rse) {
} catch (IllegalArgumentException iae) {
}
```

static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable)

- Open (and possibly create) a record store that can be shared with other MIDlet suites
- Since MIDP 2.0

```
String name = "cache";
RecordStore rs;
int authmode = RecordStore.AUTHMODE_ANY;
try {
        rs = RecordStore.openRecordStore(name, true, authmode, true);
} catch (RecordStoreException rsex) {
        rsex.printStackTrace();
}
```

static RecordStore openRecordStore(String recordStoreName, String vendorName, String suiteName)

- Open a record store associated with the named MIDlet suite
- Since MIDP 2.0

```
RecordStore rs;
try {
    rs = RecordStore.openRecordStore("cache","Youssef␣RIDENE","CoursM2TI");
} catch (RecordStoreException rsex) {
    rsex.printStackTrace();
}
```

- closeRecordStore()
- deleteRecordStore(String recordStoreName)
- getName()
- getNumRecords()
- getSize()
- getSizeAvailable()
- getVersion()
- listRecordStores()
- setMode()
  - AUTHMODE_PRIVATE
  - AUTHMODE_ANY

- MIDlet-Data-Size
- getSizeAvailable()

```
public static int recordStoreSize (){
        int rmsSize = 0;
        RecordStore rs = null;
        try{
            rs = RecordStore.openRecordStore ("RMS",true);
            rmsSize = rs.getSizeAvailable ()/1024;
        }catch (Exception e){
        }
        return rmsSize;
}
```

- A record is an array of bytes
- Each record has a unique identification number in the record store

```
RecordStore rs = null;
byte[] recordData = "The Winner".getBytes();
int id;

try {
    rs = RecordStore.openRecordStore("shared-cache","YR","CoursM2TI");
} catch (RecordStoreException ex) {
    ex.printStackTrace();
}


try {
    id = rs.addRecord(recordData, 0, recordData.length);
} catch (RecordStoreNotOpenException ex) {
    ex.printStackTrace();
} catch (RecordStoreException ex) {
    ex.printStackTrace();
}
```

- setRecord(int recordId, byte[] newData, int offset, int numBytes)
- deleteRecord(int recordId)
- getRecord(int recordId, byte[] buffer, int offset)
- getRecordSize(int recordId)
- enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)
  - hasNextElement()
  - hasPreviousElement()
  - nextRecord()
  - previousRecord()
  - nextRecordId()

- Open a RecordStore
- Write x Record(String)
- Read and display all the records
- Close the RecordStore
- Delete the Record Store

```
public RecordStore rs;
public static final String REC = "cache";

public void openRecStore() {
  try {
      rs = RecordStore.openRecordStore(REC, true);
  } catch (RecordStoreException ex) {
      ex.printStackTrace();
  }
}
```

```
public void writeRecord(String data) {
  try {
    rs.addRecord(data.getBytes(),0,data.length());
  }catch (RecordStoreNotOpenException ex) {
    ex.printStackTrace();
  }catch (RecordStoreException ex) {
    ex.printStackTrace();
  }
}
```

```
public void readRecords() {
  byte[] record;

  try {
    for (int i = 1; i <= rs.getNumRecords(); i++) {
      record = new byte[rs.getRecordSize(i)];
      record = rs.getRecord(i);
      System.out.println("Recrod␣"+i+":␣"+new String(record,0,record.length));
    }
  } catch (RecordStoreNotOpenException ex) {
      ex.printStackTrace();
  } catch (InvalidRecordIDException ex) {
      ex.printStackTrace();
  } catch (RecordStoreException ex) {
      ex.printStackTrace();
  }
}
```

```
public void closeRecStore() {
    try {
        rs.closeRecordStore();
    } catch (RecordStoreNotOpenException ex) {
        ex.printStackTrace();
    } catch (RecordStoreException ex) {
        ex.printStackTrace();
    }
}
```

```
public void deleteRecStore() {
    if (RecordStore.listRecordStores() != null) {
        try {
            RecordStore.deleteRecordStore(REC);
        } catch (RecordStoreException ex) {
            ex.printStackTrace();
        }
    }
}
```

Use of I/O streams. The steps are as follows :

- Allocate streams
- Write the data
- Flush the stream
- Transfer the stream data into an array
- Write the array to the record store
- Close the streams

```
ByteArrayOutputStream baos;
DataOutputStream dos;
byte[] record;

public void writeStream(boolean b, int i, String s) throws IOException {
    //byte array to write the data in
    baos = new ByteArrayOutputStream();
    dos = new DataOutputStream(baos);

    // Write Java data types
    dos.writeBoolean(b);
    dos.writeInt(i);
    dos.writeUTF(s);

    // Clear any buffered data
    dos.flush();

    // Get stream data into byte array and write record
    record = baos.toByteArray();
    try {
        rs.addRecord(record, 0, record.length);
    } catch (RecordStoreNotOpenException ex) {
        ex.printStackTrace();
    } catch (RecordStoreException ex) {
        ex.printStackTrace();
    }

    baos.reset();
    baos.close();
    dos.close();
```

```
public void readStream() throws IOException {
    byte[] record = new byte[50];
    ByteArrayInputStream bais = new ByteArrayInputStream(record);
    DataInputStream dis = new DataInputStream(bais);

    try {
        for (int i = 1; i <= rs.getNumRecords(); i++) {
            rs.getRecord(i,record,0);
        }

        System.out.println("b= " + dis.readBoolean() + "\n"+
                           "i= " + dis.readInt() + "\n"+
                           "s= " + dis.readUTF());

        dis.close();
        bais.close();
    } catch (RecordStoreException ex) {
        ex.printStackTrace();
    }
}
```

public int compare(byte[] rec1, byte[] rec2)

- RecordComparator.PRECEDES
- RecordComparator.EQUIVALENT
- RecordComparator.FOLLOWS

RecordEnumeration

```
rs = RecordStore.openRecordStore(REC, true);
...
RecordEnumeration re = rs.enumerateRecords(null, null, false);

while (re.hasNextElement()){
    String str = new String(re.nextRecord());
    ...
}
```

for loop

```
byte[] recData = new byte[5];
int len;

for (int i = 1; i <= rs.getNumRecords(); i++){
    if (rs.getRecordSize(i) > recData.length)
        recData = new byte[rs.getRecordSize(i)];

    len = rs.getRecord(i, recData, 0);
}
```

for loop vs. RecordEnumeration

- RecordStore with 3 records : ID = 1, ID = 2, ID = 3
- We delete record ID = 2 –> RecordStore (1,3)
- Why RecordEnumeration is important ?

boolean matches(byte[] candidate)

- true if the candidate matches the implemented criterion

- void recordAdded(RecordStore recordStore, int recordId)
- void recordChanged(RecordStore recordStore, int recordId)
- void recordDeleted(RecordStore recordStore, int recordId)

- Operations on persistent memory can take a long time on some platforms
- Must synchronize Threads for writing and reading Record Stores
- http ://develo-pers.sun.com/mobility/midp/articles/databaserms/