



Every game should have a high score system to allow gamers to challenge other gamers or themselves.

Java ME use Record Management System (RMS) to allow developers to save and restore data over different application sessions. So, for creating a quick high-score system saving name of the user and points, you can use the following code.

Contents

- [1 Source code](#)
 - ◆ [1.1 Adding a new high-score](#)
 - ◆ [1.2 Retrieving top high-scores](#)
 - ◆ [1.3 How to retrieve correctly sorted scores](#)
 - ◆ [1.4 Download](#)
 - ◆ [1.5 Implementation notes](#)
- [2 Related resources](#)

Source code

Adding a new high-score

```
// We open the recordstore
RecordStore highscore = RecordStore.openRecordStore("High Score", true);

// To add a new HiScore we use a quick string comma-separated
String str = new String("John,3400");
byte [] strb = str.getBytes();
int id = highscore.addRecord(strb, 0, strb.length);

highscore.closeRecordStore();
```

Retrieving top high-scores

```
RecordStore highscore = RecordStore.openRecordStore("High Score", true);

// To read all hiscores saved on the RMS
RecordEnumeration enum = highscore.enumerateRecords(null, null, false);
int id;
byte[] record;
String str;
while (enum.hasNextElement( )) {
    id = enum.nextRecordId( );
    record = highscore.getRecord(id);
    str = new String(record);
    // Operate with str to extract name and points
}
highscore.close();
```

How to retrieve correctly sorted scores

To sort the scores from the highest to the lowest one, an option is to implement the **RecordComparator** interface, and to use it when calling the RecordStore **enumerateRecords()** method.

A basic implementation of a *RecordComparator* that sorts the scores stored is the following one:

```
class ScoresComparator implements RecordComparator
{
public int compare(byte[] arg0, byte[] arg1)
{
int score0 = 0, score1 = 0;

for(int i = 0; i < arg0.length; i++)
{
if(arg0[i] == ',')
{
= Integer.parseInt(new String(arg0, i + 1, arg0.length - i - 1));
}
}
for(int i = 0; i < arg1.length; i++)
{
if(arg1[i] == ',')
{
= Integer.parseInt(new String(arg1, i + 1, arg1.length - i - 1));
}
}

if(score0 < score1)
return RecordComparator.FOLLOWS;
else if(score0 == score1)
return RecordComparator.EQUIVALENT;
else
return RecordComparator.PRECEDES;
}
}
```

So, the previous call to `enumerateRecords()` will accordingly change:

```
RecordEnumeration enumerator = highscore.enumerateRecords(null, new ScoresComparator(), false);
```

Download

You can download a sample MIDlet showing the code presented in this article here:

[Media:HighScoreDatabaseMIDlet.zip](#)

Implementation notes

- You should use try-catch sentences to catch exceptions on the I/O operation or make a throws.
- When writing and reading data to and from RMS, you should use separate threads to avoid blocking the MIDlet main thread.

Related resources

- ["Data Handling" section on Java ME Developer's Library](#)
- [javax.microedition.rms package JavaDocs](#)