dmRecordSet Tutorial

November 2013

Contents

1 Introduction

`IDfCollection` objects: we all know and use them, and we all wish they were more functional. For example, you can't directly create an `IDfCollection` object, but it sure would be useful if you could. You can't add columns or rows to an `IDfCollection` object or even change the values they contain, but it sure would be useful if you could. How about traversing forward and backward or randomly through an `IDfCollection`'s rows, or submitting an `IDfCollection` for update? In many ways, the `IDfCollection` class is a direct encapsulation of the collection used by the Documentum API (DMCL) years ago. That's really unfortunate, because it could potentially be so much more.

A few years ago, I wrote a series of posts on my blog discussing some of the dysfunctional aspects of the `IDfCollection` class, and offered some workarounds and an alternative[1]. My alternative to the `IDfCollection` class is the `dmRecordSet` class. This paper offers a practical guide for using the `dmRecordSet` class as a replacement for `IDfCollections` in your DFC programming.

Two primary shortcomings of the `IDfCollection` class drove me to devise and develop the `dmRecordSet` class.

1. Once a query runs, there is no way to determine the size of the result set. It is often helpful to know ahead of time and present to a user (or log file) the number of results a query produced. There is really only one way to accomplish that, as illustrated in the code below.

```
dql = "select r_object_id from dm_document where
   folder('/Temp', descend)";

q = new DfQuery();
q.setDQL(dql);
col = q.execute(session, DfQuery.DF_READ_QUERY);

// count the rows in the collection
int cnt = 0;
while (col.next()) {
   cnt++;
}
col.close();

System.out.println("Collection = " + cnt + " rows");

// if there were results, re-run the query
if (cnt > 0) {
   col = q.execute(session, DfQuery.DF_READ_QUERY);

   // do something with the IDfCollection here
}
col.close();
```

[1] http://msroth.wordpress.com/2011/11/13/idfcollections-part-0/

As you can see, the only way to determine the number of results a query returns is to iterate through the `IDfCollection` and count them. This is a brutish and inelegant solution, and leads to the second shortcoming.

2. The second shortcoming of the `IDfCollection` class is the fact that once you have iterated through it, you cannot do it again. Your only option is to rerun the query, as illustrated in the code above. This is ridiculous and can be very expensive depending upon the query and other factors of your application. There had to be a better way.

2 The dmRecordSet Class

The `dmRecordSet` class was designed and developed to address these two shortcomings, and make a collection from Documentum a little more like a result set from a database (e.g., ADO's ResultSet); thus making it a lot more useful.

In a nutshell, the `dmRecordSet` offers:

- The ability to move forward, backward or randomly through the record set;
- The ability to reset the record set to the beginning or the end to be reprocessed;
- The ability to add rows to the record set;
- The ability to determine the number of rows in the record set;
- The ability to determine if a record set is empty; and
- The ability to easily retrieve column definitions.

2.1 Public Methods

The public method signatures are:

```
•  public dmRecordSet(IDfCollection col) throws DfException
•  public void addRow(IDfTypedObject row)
•  public void addRows(ArrayList<IDfTypedObject> rows)
•  public int getColumnCount()
•  public ArrayList<IDfAttr> getColumnDefs()
•  public IDfTypedObject getCurrentRow()
•  public int getCurrentRowNumber()
•  public IDfTypedObject getFirstRow()
•  public IDfTypedObject getLastRow()
•  public IDfTypedObject getNextRow() throws Exception
•  public IDfTypedObject getPreviousRow() throws Exception
•  public List<IDfTypedObject> getRecordSetAsList()
•  public Set<IDfTypedObject> getRecordSetAsSet()
•  public String getRecordSetInfo()
•  public IDfTypedObject getRow(int rowNumber) throws Exception
•  public int getRowCount()
•  public static String getVersion()
```

- `public boolean hasNext()`
- `public boolean hasPrevious()`
- `public boolean isBOF()`
- `public boolean isEmpty()`
- `public boolean isEOF()`
- `public void resetToBeginning()`
- `public void resetToEnd()`

Hopefully the method names are descriptive enough to indicate their function.

2.2 The Role of TypedObjects

It is important to understand the role and use of the `IDfTypedObject` in the `dmRecordSet` class. `IDfTypedObjects` are non-persisted objects in the DFC. This means, they do not have a materialized equivalent in the Documentum repository and therefore lack many of the attributes and methods you expect from `IDfSysObjects`. In the `dmRecordSet` class, they represent rows in the record set. Each variable selected in the DQL query (e.g., `r_object_id` and `object_name` here), can be accessed from the `IDfTypedObject` using the `getString()`[2] method. Do not use `IDfTypedObject.getObjectId()` to obtain the `r_object_id` of this object. You will receive the `r_object_id` of the `IDfTypedObject`, which will be `0000000000000000`, and not the `r_object_id` of the object returned by the query.

3 Basic Use

This section briefly describes the basic use of the `dmRecordSet` class. Here you will find, mostly by way of example code, how to instantiate the class, move forward and backward through the records, determine properties of the record set, and how to process records in the set.

3.1 Instantiation

A new `dmRecordSet` object is instantiated by passing an `IDfCollection` object to its constructor. For example:

```
IDfCollection col = null;
String dql = "select r_object_id, object_name from dm_document where
   folder('/Temp',descend)";
IDfQuery q = new DfQuery();
q.setDQL(dql);
col = q.execute(session, DfQuery.DF_READ_QUERY);

// get record set
dmRecordSet dmRS = new dmRecordSet(col);
```

[2] Or `getInteger(), or getBoolean(), or getTime(), or getID(), or getDouble()` as appropriate for the attribute type.

The dmRecordSet object reads the content of the IDfCollection object into its internal structures and closes the IDfCollection object (IDfCollection.close()).

3.2 Test for Empty Set and Get Row Count

Once instantiated, to determine if the dmRecordSet is empty, or how many rows it contains is simple:

```
if (dmRS.isEmpty()) {
   System.out.println("dmRecordSet is empty");
} else {
   System.out.println("dmRecordSet is NOT empty");
   System.out.println("record count = " + dmRS.getRowCount());
}
```

3.3 Moving Forward, Backward, and Randomly

Navigating a dmRecordSet object follows the pattern modeled in the following code. The dmRecordSet class has basic navigation methods to move forward, backward, or randomly through the record set.

```
// iterate forward
tObj = dmRS.getFirstRow();  // move to BOF if not already there
while (dmRS.hasNext()) {
   IDfTypedObject tObj = dmRS.getNextRow();
   System.out.print(tObj.getString("r_object_id") + "\t");
   System.out.println(tObj.getString("object_name");
}

//iterate backward
tObj = dmRS.getLastRow();  // move to EOF
while (dmRS.hasPrevious()) {
   IDfTypedObject tObj = dmRS.getPreviousRow();
   System.out.print(tObj.getString("r_object_id") + "\t");
   System.out.println(tObj.getString("object_name");

// get random row
Random generator = new Random();
int r = generator.nextInt(dmRS.getRowCount());

System.out.println("Random row " + r);
IDfTypedObject tObj = dmRS.getRow(r);
System.out.print(tObj.getString("r_object_id") + "\t");
System.out.println(tObj.getString("object_name");
```

3.4 Processing a dmRecordSet

Section 3.3 presented the basic idea of navigating a dmRecordSet and doing something with its contents. This section will take that basic idea a little further and demonstrate how to process a dmRecordSet that you don't know anything about (for example, processing a user-entered query), and how to access the dm_sysobjects represented by the rows of the record set.

```
// get all of the column names of the record set
ArrayList<IDfAttr> cols = dmRS.getColumnDefs();

// print col names as headers
for (IDfAttr a : cols) {
   System.out.print(a.getName() + "\t");
}
System.out.println("object type");

// print record set content under each column heading
while (dmRS.hasNext()) {
   tObj = dmRS.getNextRow();
   for (IDfAttr a : cols) {
      System.out.print(tObj.getString(a.getName()) + "\t");
   }
   System.out.println();
}
```

Take not of the use of the `IDfTypedObject`, `tObj`, in the above code. Remember this object represents a row in the record set. Also notice the use of the array of column names (`cols`), and how it is used to retrieve values from the record set.

3.5 Obtaining an IDfSysObject from a dmRecordSet

Sometimes you need more than just the rows and columns of a query result. Sometimes you need the actual `dm_sysobject` represented by the rows and columns. It is simple to request the `dm_sysobject` from the `IDfSession` associated with the `IDfTypedObject` in the `dmRecordSet`. Once obtained, you can check it out and manipulate it as needed.

```
while (dmRS.hasNext()) {
   tObj = dmRS.getNextRow();

// assumes r_object_id was included in the query
   IDfSysObject sObj = (IDfSysObject) tObj.getSession().getObject
      (tObj.getId("r_object_id"));
   sObj.checkout();
   if (sObj.isCheckedOut())
      System.out.println("checked out: " + sObj.getObjectName());
   sObj.cancelCheckout();
}
```

4 More Advanced Uses

Section3 demonstrated the basics you need to know to get started using the `dmRecordSet` class in your DFC code. I suspect the examples given there cover the vast majority use cases for most users; however, there are some additional, not-so-common, yet interesting things we can still do with the `dmRecordSet` class.

4.1 Testing dmRecordSet Boundaries

The `dmRecordSet` will throw Exceptions if you try to access records outside of its boundaries. For example:

```
try {
   tObj = dmRS.getRow(-100);
   for (IDfAttr a : cols) {
      System.out.print(tObj.getString(a.getName()) + "\t");
   }
   System.out.println();
} catch (Exception e) {
   System.out.println("\t" + e.getMessage());
}
```

The code above throws an exception because row -100 does not exist in the record set (obviously). You can use a simple `try-catch` model when iterating over a `dmRecordSet` to catch unexpected boundary exceptions. The same model works when trying to access rows past the end of the record set also, as demonstrated below.

```
try {
   tObj = dmRS.getRow(dmRS.getRowCount() + 100);
   for (IDfAttr a : cols) {
      System.out.print(tObj.getString(a.getName()) + "\t");
   }
   System.out.println();
} catch (Exception e) {
   System.out.println("\t" + e.getMessage());
}
```

4.2 dmRecordSet as a Java List

You can obtain the record set as a Java `List<IDfTypedObject>` if you prefer.

```
List<IDfTypedObject> list = dmRS.getRecordSetAsList();
for (IDfTypedObject t : list) {
   System.out.println(t.getString("r_object_id"));
}
```

4.3 dmRecordSet as a Java Set

You can also obtain the record set as a Java `Set<IDftypedObject>`. One nice feature of the `Set` is that it only includes unique items.

```
Set<IDfTypedObject> set = dmRS.getRecordSetAsSet();
for (IDfTypedObject t : set) {
   System.out.println(t.getString("r_object_id"));
}
```

4.4 dmRecordSet Stats

You can obtain some basic status info from each dmRecordSet by using the status() method, as demonstrated below.

```
System.out.println("\nFinal record set stats:");
System.out.println(dmRS.getRecordSetInfo());
```

The status information returned includes:

- `dmRecordSet` version number,
- total row count,
- current row,
- total column count,
- names of columns,
- BOF indicator,
- EOF indicator.

4.5 Insert into a dmRecordSet

You can incrementally build a `dmRecordSet` by adding `IDfTypedObjects` to it either one at a time or as an `ArrayList<IDftypedObjects>`. The `addRow()` and `addRows()` methods validate the rows to ensure the columns match the record set columns before adding the rows.

```
dmRS.addRow(dmRS.getLastRow());

ArrayList<IDfTypedObject> Temp = new ArrayList<IDfTypedObject>();
   for (int j = 0; j < r; j++) {
      Temp.add(dmRS.getRow(j));
   }

dmRS.addRows(Temp);
```

5 Download

The `dmRecordSet` is distributed as a ZIP archive, and can be downloaded from: https://app.box.com/s/xmajl7jfu1rtnng7vjiu

The content of the ZIP file is:

• dmRecordSet JAR file	`dmRecordSet.jar`
• dmRecordSet tutorial	`dmRecordSet_Tutorial.pdf`
• dmRecordSet code examples	`dmRecordSet_examples.java`
• dmRecordSet Javadoc	`/javadoc`

6 Licensing

This work is licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License.

Basically you to use `dmRecordSet` in just about any way you wish – commercially and non-commercially – as long as you credit me, and don't change the source code. If you require changes to the source code please contact me.