dmRecordSet Tutorial

August 2013

© M. Scott Roth / [msroth.wordpress.com](http://msroth.wordpress.com)

# 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 (winter, 2011), 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]](#footnote-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 ran, there was 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 was 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++;

}

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

}

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.

1. 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.

# The dmRecordSet Class

In a nutshell, what this class 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 or not; and
* The ability to easily retrieve column definitions.

The public method signatures are:

* public dmRecordSet(IDfCollection col) throws DfException
* public int getRowCount()
* public int getColumnCount()
* public ArrayList<IDfAttr> getColumnDefs()
* public boolean isBOF()
* public boolean isEOF()
* public boolean isEmpty()
* public boolean hasNext()
* public IDfTypedObject getNextRow() throws Exception
* public boolean hasPrevious()
* public IDfTypedObject getPreviousRow() throws Exception
* public IDfTypedObject getFirstRow()
* public IDfTypedObject getLastRow()
* public IDfTypedObject getRow(int rowNumber) throws Exception
* public IDfTypedObject getCurrentRow()
* public int getCurrentRowNumber()
* public void resetToBeginning()
* public void resetToEnd()

<http://msroth.wordpress.com/2011/12/19/idfcollections-part-v/>

# Basic Use of the dmRecordSet Class

## Instantiation

A new dmRecordSet object is instantiated by passing an IDfCollection object to its constructor.

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);

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

## Test for Empty Set and Get Row Count

Once instantiated, determining 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());

}

## Moving Forward, Backward, and Randomly

Processing the dmRecordSet object after it has been instantiated follows the pattern described in the code below. The dmRecordSet class has basic iterator methods to help you process a record set, forward, backward, or randomly.

// iterate forward

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();

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");

It is important to understand the role and use of the IDfTypedObject in this code. 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 this context, 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]](#footnote-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.

## Processing a Record Set

ArrayList<IDfAttr> cols = dmRS.getColumnDefs();

while (dmRS.hasNext()) {

tObj = dmRS.getNextRow();

for (IDfAttr a : cols) {

System.out.print(tObj.getString(a.getName()) + "\t");

}

System.out.println();

}

# More Advanced Uses

Testing boundaries

Recursion

# Download

# Licensing

Which?

1. <http://msroth.wordpress.com/2011/11/13/idfcollections-part-0/> [↑](#footnote-ref-1)
2. Or getInteger(), or getBoolean(), or getTime(), or getID(), or getDouble() as appropriate for the attribute type. [↑](#footnote-ref-2)