EMC White Paper

EMC² where information lives

DFC Session Management

Srinivas Jakkula

Abstract: This document highlights the nuances of DFC session management from the client application perspective.

Date 3/11/2007

EMC² | documentum

DFC Session and Session Manager Overview

DFC Session gives access to specific user to a specific repository. This means that a DFC session represents a client's connection to a repository with specified access credentials. A session object is an instance of IDfSession Interface. Almost every functionality that you access through DFC requires a repository (DMCL) session and DFC session contains repository session. DFC clients get an access to session object by using Session Manager.

Session Manager manages sessions with one or more repositories. Session manager is used by the client applications to retrieve sessions and release them back when they are done using the session. SessionManager also maintains a pool of DMCL sessions and allocates them to the DFC sessions when requested. DFC Session should be obtained through Session Manager rather than directly getting them using IDfClient object.

Session Manager provides methods to enable the DFC clients to:

- Manage Identities/access credentials.
- Authenticate user access.
- Acquire and release sessions.
- Process Transactions
- Authenticate Users
- Obtain session statistics
- Catch checked and unchecked exceptions.

Session Manager and Session Pooling

Session Manager provides DMCL (repository) session pooling capability to provide efficient handling of the sessions as there is considerable overhead in creating and destroying repository sessions. Session Manager maintains a pool of DMCL sessions. When you release a session that you obtained from the session manager, the Session Manager waits a short while before disconnecting it. If you ask the Session Manager to get you a session with the same repository before it has disconnected the old one, it gives you back the old one, saving the substantial overhead of authenticating your login information and creating a new session. If the session manager has disconnected the old session, or if you ask it to give you a new session or a session with a different repository, the session manager silently obtains a new session for you.

Session manager also works properly if the connection pooling is turned off but is inefficient.
You can get sessions from Session manager by using either getSession or newSession method on the IDfSessionManager interface. getSession returns an existing DMCL session from the connection pool if one exists or creates a new session is no free DMCL sessions are available. newSession creates a new DMCL session and returns irrespective of whether there is a free connection or not. Shared session returned by the getSession method reduce the resources your application uses. But at the same time one has to be careful as two threads referencing the same repository object obtained from the same session get reference to the same object and modifying it one thread will cause the cause the object to change in the other thread even though nothing has been modified in the second thread.

It should be noted that DFC pooling is different from DMCL pooling and release of DfSession does not invoke dmcl disconnect immediately but after certain period of inactivity.

Transaction Handling

Because the session manager handles sessions with more than one repository, it supports a transaction mechanism that encompasses interactions with more than one repository. It still relies on the transaction mechanism of each repository's underlying relational database. Nested transactions are not supported.

Session manager transactions do not use a two-phase commit algorithm, and they rely on the transaction facilities of the underlying databases. As a result, a multi-repository transaction can fail after the system has committed the transaction in one of the repositories. For example, if there are three repositories, and the first two commit successfully, but the third does not, the session manager cannot undo the already committed transactions on the first two repositories.

Session Transaction and Session Manager Transaction are different. IDfSession interface has beginTrans method that corresponds to the explicit database transaction. Note that any changes that you save or check in are not committed to a repository until you call the commitTrans method. You can cancel the database transaction by calling the abortTrans method.

IDfSessionManager has beginTransaction method to start the Session manager transactions. This method starts a new managed transaction. Only sessions obtained after the call to beginTransaction method will be able to participate in the transaction. Session objects obtained before beginTransaction will not be able to participate in the transaction. You can commit the transaction by using the commitTransaction method or abort the transaction by calling the abortTransaction method.

Lifecycle of DFC Client Application

Although every DFC Client Application varies in the functionality and complexity depending upon the business requirement, there are some steps that every DFC client typically goes through to achieve a particular functionality with the repository. The following is overview of a typical DFC client functionality in handling sessions.

1. Obtaining a DFC Client object.
2. Getting a new instance of Session Manager
3. Registering user access credentials for accessing the repository with the Session Manager.
4. Getting a DFC session to work with the repository.
5. Performing business functionality with the repository.
6. Releasing the session back to the Session Manager.
7. Destroying or abandoning the Session Manager.

Session Leaks and Diagnosing leaks

Session leaks can result when a caller does not return a session back to the pool. If a session is not released back to the pool, the session manager cannot reuse it. Thus, if a new request for a session comes in, session manager will have to create a new session. If this goes on for an extended period of time, a large number of unusable sessions will be created and eventually DFC will run out of sessions. This makes it important to find the source of session leaks.

As mentioned earlier, every IDfSessionManager#getSession(...) should be followed by IDfSessionManager#release().An IDfSession object should not be stored in a place where its release cannot be guaranteed. Thus, its not a good idea to store it as a class member variable or store it in some sort of a cache.

To diagnose session leaks by making following changes.

- Open the *$DOCUMENTUM\config\dfc.properties* file. On our machine this file was in the *c:\Documentum\config* folder.
- To this file add the following line - dfc.resources.diagnostics.enabled=true .
- Save and close the dfc.properties file.
- Open log4j.properties file located under $DOCUMENTUM\config directory and add the following lines at the end. These entries with direct the logs for session leaks into sessLeakDetector.log.

```
For DFC 5.2.5 SPx
=================

log4j.logger.com.documentum.fc.client.DfSessionLeakChecker=
ERROR,SESS_LEAK_DETECT
log4j.appender.SESS_LEAK_DETECT=org.apache.log4j.RollingFil
eAppender
log4j.appender.SESS_LEAK_DETECT.File=C\:/Documentum/logs/se
ssLeakDetector.log
log4j.appender.SESS_LEAK_DETECT.MaxFileSize=100MB
log4j.appender.SESS_LEAK_DETECT.MaxBackupIndex=5
log4j.appender.SESS_LEAK_DETECT.layout=org.apache.log4j.Pat
ternLayout
log4j.appender.SESS_LEAK_DETECT.layout.ConversionPattern=%d
{ABSOLUTE} [%t] %m%n

For DFC 5.3 and Higher
======================

log4j.logger.com.documentum.fc.client.DfSessionManagerPool=
DEBUG,SESS_LEAK_DETECT
log4j.logger.com.documentum.fc.client.DfSessionmanager=DEBU
G,SESS_LEAK_DETECT
log4j.logger.com.documentum.fc.client.DfDisposableCollectio
n=DEBUG,SESS_LEAK_DETECT
log4j.appender.SESS_LEAK_DETECT=org.apache.log4j.RollingFil
eAppender
log4j.appender.SESS_LEAK_DETECT.File=C\:/Documentum/logs/se
ssLeakDetector.log
log4j.appender.SESS_LEAK_DETECT.MaxFileSize=100MB
log4j.appender.SESS_LEAK_DETECT.MaxBackupIndex=5
log4j.appender.SESS_LEAK_DETECT.layout=org.apache.log4j.Pat
ternLayout
log4j.appender.SESS_LEAK_DETECT.layout.ConversionPattern=%d
{ABSOLUTE} [%t] %m%n
```

Below is a sample session leak which gives you. The stack trace will tell you the exact class from which the leak originated

```
        15:09:38,803 [Resource Housekeeper] DFC_BOF_SESSION_LEAK|
Unreleased session found in
```

```
finalize "docbase=wdk52no, refCounter=2, transFlag=false  Session
=com.documentum.fc.client.DfSession@19ee8a".
com.documentum.fc.client.DfSessionLeakException
        at
com.documentum.fc.client.DfSessionLeakChecker.<init>(DfSessionManager.j
ava:1186)
        at
com.documentum.fc.client.DfSessionManager.createRequiredSessionObject(D
fSessionManager.java:513)
        at
com.documentum.fc.client.DfSessionManager.getSession(DfSessionManager.j
ava:507)
        at
com.documentum.fc.client.DfSessionManager.getSession(DfSessionManager.j
ava:283)
        at
com.documentum.custom.test.TestDFCDiag.onClickHello(TestDFCDiag.java:41
)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.ja
va:39)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccesso
rImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:324)
        at
com.documentum.web.form.FormProcessor.doInvokeMethod(FormProcessor.java
:1168)
        at
com.documentum.web.form.FormProcessor.invokeMethod(FormProcessor.java:9
38)
        at
com.documentum.web.form.FormProcessor.fireActionEvent(FormProcessor.jav
a:769)
        at
com.documentum.web.form.RecallOperation.execute(RecallOperation.java:98
)
        at
com.documentum.web.form.FormProcessor.openForm(FormProcessor.java:136)
        at
com.documentum.web.form.WebformTag.doStartTag(WebformTag.java:125)
        at
org.apache.jsp.testDFCDiag_jsp._jspx_meth_dmf_webform_0(testDFCDiag_jsp
.java:69)
        at
org.apache.jsp.testDFCDiag_jsp._jspService(testDFCDiag_jsp.java:46)
        at
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:137)
       ......
```

This mode is used to diagnose session related issues. It can impact performance and should be preferably used on development/test systems instead of production systems.

SessionListener

In DFC, you can trap session creation and destruction by writing a session listener class that implements IDfSessionManagerEventListener. IDfSessionManagerEventListener interface has two methods onSessionCreate and onSessionDestroy that have to be implemented as what you intend to do by trapping the session creation and destruction.

Once you have the session listener class, you need to set this class as listener to the SessionManager by using IDfSessionManager.setListener(…) method.

Below is a simple implementation of the session listener that prints out at every session creation and destruction.

```
static class SessionListenerImpl implements IDfSessionManagerEventListener
  {
      public void onSessionCreate(IDfSession sess) throws DfException {
          System.out.println("Session created: "  + sess.getSessionId());

      }

      public void onSessionDestroy(IDfSession sess) throws DfException {
          System.out.println("Session Destroyed: " + sess.getSessionId());
      }

  }
```

Orphaned Objects

Orphaned objects are objects obtained from a session that has already been released back to the DFC session manager. Support for detection of orphaned objects was added in DFC 5.3. Orphaned objects can be detected by enabling DFC session diagnostics. This will become more complex if there are multithread applications where objects are passed around. Keeping this in mind, it is advised no to the object instance after the session is released and if the object needs to be passed around, you should be passing the IDfId and then use this id to get the object again by using IDfSession.getObjectById(…).

DFC sessions and WDK Applications

In WDK Applications, there will be only one instance of IDfSessionManager. You can get reference to IDfSessionManager by using SessionManagerHttpBinding.getSessionManager() method. It is not advisable to get a new SessionManager in WDK applications. Also whenever you need to work with a IDfSession in WDK applications use the getDfSession method of the Component class and not get a session from session manager. This way the release of sessions is handled by the component and you do not have to explicitly release the session. The objects retrieved from a session should be short lived and should be within the scope of one request/response. SysObjects should not be persisted for more than a request/response and the underlying session that created the SysObject might have been disconnected. Instead you use the strategy mentioned in ‘Orphaned Objects’ section i.e. to hold on to object id and retrieve the SysObject whenever you need it.

How Do I Section

1. How to get a DFC session and release it back to the pool.

The following code snippet give you the basic steps needed to get a session, use it and release it back to the pool.

```
//create Client objects
 IDfClientX clientx = new DfClientX();
 IDfClient client = clientx.getLocalClient();
```

```
//create a Session Manager object
IDfSessionManager sMgr = client.newSessionManager();

//create an IDfLoginInfo object for user creddentials
IDfLoginInfo loginInfoObj = clientx.getLoginInfo();
loginInfoObj.setUser(<user>);
loginInfoObj.setPassword(<pass>);
loginInfoObj.setDomain(null);

//bind the Session Manager to the login info
sMgr.setIdentity(<docbase>, loginInfoObj);


IDfSession session = null;

try {
        //get the IDFsession instance by using getSession or newSession
        Session = sMgr.getSession(<docbase>);

        //user the session to perform repository functions
        …….
        …….

}finally{
        //release the session
        sMgr.release(session);
}
```

2. How do I assign the same user access credentials to all the repositories?

You can assign the same user credentials to all the repositories in one step instead of using IDfsessionManager setIdentity method for each of the repositories by proving a '*' instead of the repository name in the setIdentity method.

```
sMgr.setIdentity( * , loginInfoObj );
```

3. How do I get a new new/fresh session from the repository instead of getting one from the session pool?

You can get a new session from the repository by using IDfSessionManager's newSession method instead of getSession method.

4. How do I turn off Connection Pooling in Session Manager?

You can turn of connection pooling so that every time a session is requested, a new repository session is created by setting the 'connect_pooling_enabled' property in the dmcl.ini file to 'F'.

5. How do I make DFC to disconnect the DMCL session immediately after releasing the session?

You can achieve this by including 'DebugSessionManager' option in the JVM startup of the client program. This causes the session manager to disconnect sessions as soon as you

release them, or (to simulate server timeouts) after a short interval even if you don't release them. As the option implies, it is a Debug parameter and it is not recommended to use this in production applications as it can affect performance.

Questions 6-9 apply to DMCL sessions and not to IDfSessionManager.

6. How do I configure the timeout for inactive DMCL session?

The default timeout for an inactive dmcl session is 5 minutes. To change this, modify the 'client_session_timeout' key in the server.ini file. The server needs to be restarted for this change to take affect.

7. How do I configure that each DMCL session is only used for maximum n number of times?

'connect_recycle_interval' parameter in dmcl.ini controls how many times a repository session can be reused. The limit prevents a single session process from becoming too big. The default value for this is 1000.

8. How do I configure the maximum number of sessions a client can open with the repository?

By default only 10 sessions are allowed per client. To change this, modify the max_session_count key in the dmcl.ini file. The client needs to be restarted for this to take affect.

9. How do I configure the maximum number of concurrent sessions allowed for all the clients combined for a repository?

By default maximum concurrent sessions on the server are limited to 100. To change this, modify the concurrent_sessions key in the server.ini file. The server needs to be started for this to take affect.

10. How do I handle transactions?

You can handle transaction by using the transaction capability of the Session manager. The following code snippet gives you the template to use transactions. Remember that only the sessions obtained after the beginTransaction will participate in the transaction.

```
try {
        //start the transaction
        sMgr.beginTransaction();

        //get the session
        session = sMgr.getSession(docbase);

        //perform actions
        …….

        //commit the transaction
        sMgr.commitTransaction();

}catch(exception ex){
        //handle exception
        ……
```

```
            //abort the transaction
            sMgr.abortTransaction();
}finally{
            //release the session
            sMgr.release(session);
}
```

11. How do I enable session diagnostics?

You can enable session diagnostics by adding 'dfc.resources.diagnostics.enabled=true' to the dfc.properties file and possibly directing the logs to a separate file by configuring an appender for session diagnostics as described in the session diagnostics section.

12. Are DFC Sessions thread-safe?

From DFC 4.2.7 onwards, all the calls on IDfSession and IDfSessionManager are synchronized. Hence calls on the IDfSession and IDfSessionManager should be thread safe.

13. How do I distinguish whether a session is alive or timed out in DFC?

Currently there is no way to tell if a DFC session is active or if the session expired. Even though the current session expires by the server, new session shall be created if the DFC application tries to query using the original session identifier (q0, q1 etc).

IDfSession.isConnected method returns always true whether the session is alive or timed out unless the session is disconnected by IDfSession.disconnect().

Challenge Questions

1) What are the requirements for a DfSession to participate in a SessionManager transaction?
2) How do you write a custom code that depends on when the a new session is created or destroyed.?
3) How to make DFC disconnect the dmcl connection immediately after release?
4) How to get a session for another user when logged in as a different user?
5) How do get a session in WDK application?
6) How to know about the number of active sessions with SessionManager?

About EMC

EMC Corporation (NYSE: EMC) is the world leader in information storage systems, software, networks, and services, providing automated networked storage solutions to help organizations get the maximum value from their information, at the lowest total cost, across every point in the information lifecycle. Information about EMC's products and services can be found at www.EMC.com

About EMC Documentum Software

EMC Documentum software includes enterprise content management solutions that enable organizations to unite teams, content, and associated business processes. With a single platform, EMC Documentum

software enables people to collaboratively create, manage, deliver, and archive the content that drives business operations, from documents and discussions to e-mail, Web pages, records, and rich media. The Documentum enterprise content management platform makes it possible for companies to distribute all of this content in multiple languages, across internal and external systems, applications, and user communities. As a result of deploying Documentum, thousands of the world's most successful organizations are harnessing corporate knowledge, accelerating time to market, increasing customer satisfaction, enhancing supply chain efficiencies, reducing operating costs, and improving their overall competitive advantage.

For more information about Documentum enterprise content management, visit www.emc.com/documentum or call **800.607.9546** (outside the U.S.: +1.925.600.5802).