Joern Klauke works as a software developer for SAP on DB2 for Linux, UNIX, and Windows at the IBM Research and Development Lab in Boeblingen (Germany). He has five years of experience with IBM and DB2, assisting customers with best practices, problem analysis and troubleshooting. He holds a degree in Computer Science from the Martin-Luther-University of Halle (Germany).

Part 1 – Introduction to the Scripted Interface for DB2 Advanced Copy Services

The DB2 Advanced Copy Services (DB2 ACS) support taking snapshots for backup purposes in DB2 for LUW databases. You can use the DB2 ACS API through libraries implemented by your storage hardware vendors, which up to now, only some vendors do. Alternatively, you can implement this API yourself, which, however, involves a high effort.

DB2 10.5 introduces a new feature called Scripted Interface for DB2 Advanced Copy Services. It allows you to implement shell scripts instead of C-libraries. These scripts can use the tools provided by the storage vendors to run the snapshot operations. The Scripted Interface can be used independently from your storage hardware. Additionally, DB2 supports every storage hardware as soon as it becomes available on the market.

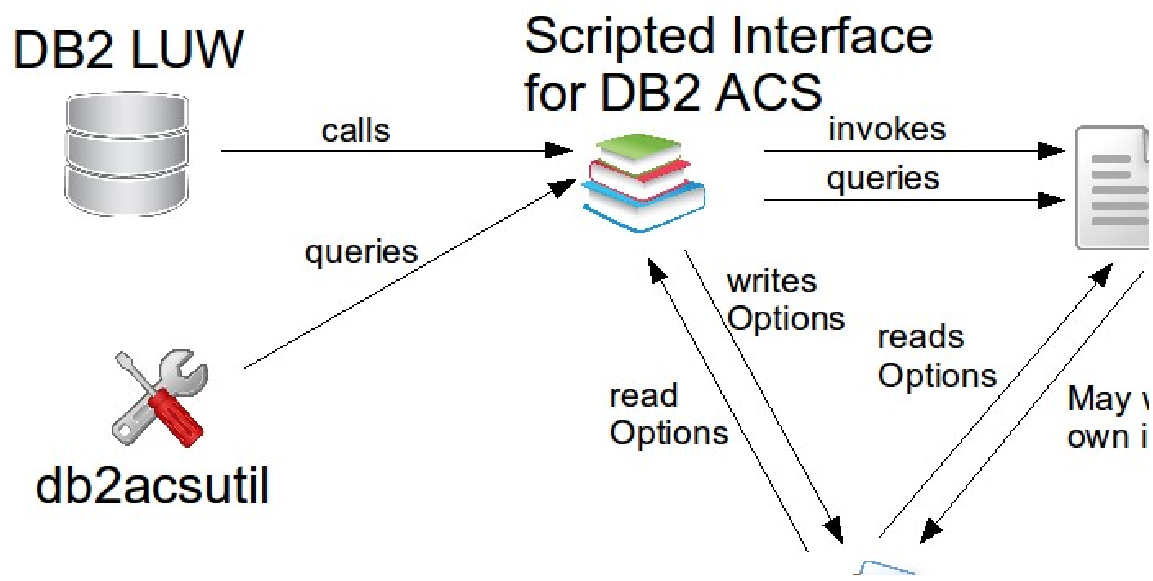
The feature supports all three architectures of DB2: enterprise server, multi-partitioned database using the database partitioning feature (DPF), and databases using pureScale. It is supported on all UNIX and Linux platforms that DB2 is certified on.

In this series we will provide an introduction to the Scripted Interface feature and present some real-life examples in the coming parts.

This first part of the series will give you an overview of the feature and describe in detail the components that it consists of. The general setup will be shown based on the example of the Sample Customer Script provided by DB2.

Introduction

The following figure shows the overall structure of the new Scripted Interface for DB2 ACS:



You can see the three important parts of the feature: the DB2 server including the Scripted Interface for DB2 ACS, the Protocol File Repository, and the Customer Script. The DB2 server triggers the snapshot with the backup command and invokes the Customer Script. The Protocol File Repository uses the Protocol Files to which DB2 writes important information that can be used by the Customer Script to run a successful snapshot, for example, the database name and the paths that have to be copied during the backup. In the following sections, we take a further look at all the components of the Scripted Interface feature and provide examples for the database JK1 in the instance db2jk1.

Protocol Files and Protocol File Repository

The Protocol File Repository is a directory in the operating system where the snapshots are taken. It can be any directory that provides the following privileges:

- The directory must exist before the command is started
- The instance owner of the instance that owns the database has to be able to read and write in this directory.
- There has to be enough space to hold the Protocol Files written during the operations.

You can use the following command to run a snapshot backup in DB2 in order to use the Scripted Interface:

```
BACKUP DATBASE JK1
USE SNAPSHOT SCRIPT '$HOME/sqllib/samples/BARVendor/libacssc.sh'
OPTIONS '/repository'
```

The following command can be run in both CLP and ADMIN_CMD:

```
CALL SYSPROC.ADMIN_CMD("BACKUP DATBASE JK1
USE SNAPSHOT SCRIPT '$HOME/sqllib/samples/BARVendor/libacssc.sh'
OPTIONS '/repository'")
```

In this command, `$HOME/sqllib/samples/BARVendor/libacssc.sh` is used as the location and the name of the customer script, that is, the Sample Customer Script provided by DB2. `/repository` is used as the directory for the Protocol File Repository. Note that you have to use the absolute path for both the script name and the repository. If you do not provide the repository, the directory where the customer script is located is used as the Protocol File Repository. If you want to provide further options to your Customer Script and you use the OPTIONS clause to do so, the first parameter is always parsed as the Protocol File Repository.

The Protocol Files serve two purposes:

- They provide information on the database that is necessary for DB2 to perform a restore. Therefore, we recommend that you back up the protocol files.
- They give information on the progress of the backup for investigation purposes, e.g. the timestamps of the start and end of each function and the commands used to invoke the script starting with "`# cmd:`"

The name of each Protocol File is generated by DB2 with the following structure:

$$\underbrace{\texttt{db2acs}}_{\text{Prefix}}\texttt{.}\underbrace{\texttt{JK1}}_{\text{DB}}\texttt{.}\underbrace{\texttt{0}}_{\text{Partition}}\texttt{.}\underbrace{\texttt{db2k1}}_{\text{Instance}}\texttt{.}\underbrace{\texttt{1366621223}}_{\text{Handle}}\texttt{.}\underbrace{\texttt{cfg}}_{\text{Suffix}}$$

Both the DB2 and the Customer Script write information to the Protocol Files but at the moment, information written by the Customer Script is not evaluated by DB2. If you want to write your own options to the Protocol Files, it would be best to use "USER" as the common prefix to avoid future collisions with new DB2 options. DB2 uses key-value-pairs for the options, separated by the equal sign ("=").

During each operation, a new Protocol File is written. The Protocol Files of the backup operations are kept in the case of success failure. Protocol Files of failed backups have to be removed manually from the repository in order to avoid issues during restore operations. Protocol Files of other operations such as delete, query, or restore operations, are only kept in case of failure. They should also be removed after investigations.

The Protocol Files are separated in sections that reflect the function calls to the DB2 ACS API. Each section starts and ends with a timestamp to give an idea on how long each call took. Additionally, the information provided by the DB2 ACS API during that function call is provided in each section. The following shows the db2ACSBeginOperation call from a Protocol File of a backup operation:

```
# ========================================================
# db2ACSBeginOperation(): BEGIN [Mon Apr 22 05:00:23 2013]
OPERATION=SNAPSHOT
# db2ACSBeginOperation(): END [Mon Apr 22 05:00:23 2013]
# ==========================================================
```

You can see the name of the function, the two timestamps, and the specified operation, in this case SNAPSHOT for snapshot backup.

For the subsequent sections, the repository `/repository` is used.

The Protocol Files of all operations start with a common section for the function db2ACSInitialize that looks as follows:

```
# ==========================================================
# db2ACSInitialize(): BEGIN [Mon Apr 22 05:00:23 2013]
EXTERNAL_SCRIPT=/home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
EXTERNAL_OPTIONS=/repository 2ndoption
DB_NAME=JK1
INSTANCE=db2jk1
DBPARTNUM=0
SIGNATURE=SQL10050
# db2ACSInitialize(): END [Mon Apr 22 05:00:23 2013]
# ==========================================================
```

As you can see, this above section already specifies from which database in which instance running on which version (SIGNATURE) the command was started. The snippet above is taken from a snapshot operation.

Detailed Look at Customer Scripts Based on the Sample Customer Script

In the following, we will describe in detail which actions are taken by the Sample Customer Script during each operation.

Each invocation looks like the following:

```
/home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
-a prepare
-c /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
/repository 2ndoption
```

The flag `-a` is followed by the action to be taken in that invocation, the flag `-c` is followed by the currently used Protocol File. The options that were given in the options clause of the backup command are appended to the script command. Each command is also written to the Protocol File by DB2.

As already mentioned, DB2 provides a Sample Customer Script to demonstrate the structure and possible usage with the name `libacssc.sh` in the instance path `$HOME/sqllib/samples/BARVendor`. The examples in the coming sections will use this script. This script handles options by parsing it as soon as it is called by the following lines:

```
while getopts a:c:o:t: OPTION
do
   case ${OPTION} in
      a) action=${OPTARG}
         ;;
      c) config=${OPTARG}
         ;;
      o) objectId=${OPTARG}
         ;;
      t) timestamp=${OPTARG}
         ;;
      \?) echo "# Unknown parameter '$1'"
   esac
done
```

The action option (-a) is used to call the corresponding function in the shell script as shown in the following snippet where the strings starting with *do* are the names of the functions:

```
case "$action" in
   prepare)
      doPrepare
      ;;
   snapshot)
      doSnapshot
      ;;
   restore)
      doRestore
      ;;
   delete)
      doDelete
      ;;
   verify)
      doVerify
      ;;
   store_metadata)
      doStoreMetaData
      ;;
   rollback)
      doRollback
      ;;
esac
```

There are actions that the Customer Script can support that occur during more than one operation. For example, the prepare action is performed during all operations, including snapshot, restore, query, and delete operations. For this purpose, the calls for these actions should have the following structure to be able to run different steps for different operations:

```
getSetting "OPERATION"
operation=$_setting
case "$operation" in
   snapshot)
      ...
```

```
        ;;
    delete)
        ...
        ;;
    restore)
        ...
        ;;
    query)
        ...
        ;;
esac
```

Each action is ended with a return code that is written to the Protocol File by DB2 with the common prefix "`RC_`". If, for example, the action prepare was terminated in the Customer Script by `exit 1,` DB2 will write the line `RC_PREPARE=4` to the Protocol File.

The Sample Customer Script contains two help functions. The function "getSetting" reads a certain option from a Protocol File. The most important line in this function is the following:

```
cmd="awk -F= '/^${1}/ { print \$2 }' $useConfig | head -1"
```

This function reads the complete Protocol File with the help of the UNIX tool *awk* and parses for the given name of the key. The value is separated by the equal sign ("="). `head -1` limits the number of returned values to the first one.

The second function "storeSetting" writes options to the Protocol File, separating them by the equal sign ("="), which makes it easier to meet this definition. It does this with the following line:

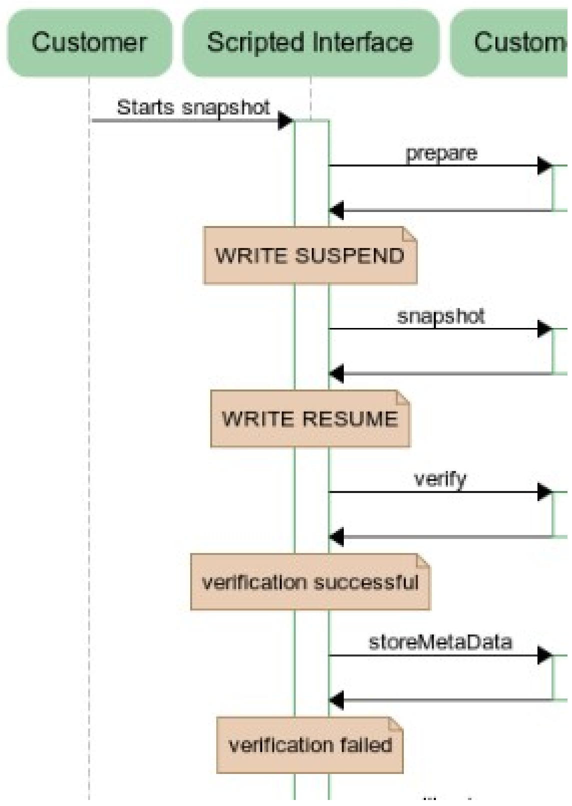
```
echo "$1=$2″
```

In general, the output of each echo command without a target used in the Customer Script is written to the Protocol Script. If you want to use another file for your debugging information, append a target to the echo commands:

```
echo "$1=$2″ >> target_file
```

Snapshots

During the snapshot operation, the Customer Script is called four times for the following actions: prepare, snapshot, verify, and - depending on the result of the verify call storemetadata or rollback. The following figure shows an overview of the actions of the snapshot operation:



When the Customer Script is called the first time the following information is already given in the Protocol File besides the information given by the db2ACSInitialize call as shown above:

```
# ========================================================
# db2ACSBeginOperation(): BEGIN [Mon Apr 22 05:00:23 2013]
OPERATION=SNAPSHOT
# db2ACSBeginOperation(): END [Mon Apr 22 05:00:23 2013]
# ========================================================
# db2ACSPartition(): BEGIN [Mon Apr 22 05:00:23 2013]
OBJ_HOST=hal9000
OBJ_OWNER=
OBJ_TYPE=SNAPSHOT
OBJ_DB2ID_LEVEL=0
OBJ_DB2ID_RELEASE=5
OBJ_DB2ID_VERSION=10
APP_OPTIONS=1100
TIMESTAMP=20130422050024
DB2BACKUP_MODE=ONLINE
DB2BACKUP_LOGS=INCLUDE
LOGPATH_PRIMARY=/home/db2jk1/db2jk1/NODE0000/SQL00001/LOGSTREAM0000/
DATAPATH_DB=/home/db2jk1/db2jk1/NODE0000/SQL00001/MEMBER0000/
DATAPATH_LOCAL_DB=/home/db2jk1/db2jk1/NODE0000/sqldbdir/
DATAPATH_DB=/home/db2jk1/db2jk1/NODE0000/SQL00001/
DATAPATH_AUTOSTORAGE=/home/db2jk1/db2jk1/NODE0000/JK1/
# db2ACSPartition(): END [Mon Apr 22 05:00:23 2013]
# ========================================================
```

The most important options start with either DATAPATH or LOGPATH; they show the paths and the log directories that have to be included in the snapshot. As you can see, the type of data path is also shown, e.g., autostorage paths (DATAPATH_AUTOSTORAGE) and database paths (DATAPATH_DB). The same is true for log directories that are specified as LOGPATH_PRIMARY (for the primary log path) and LOGPATH_MIRROR (for the mirrored log path if it exists). Of course, log paths are only given if logs have to be included in the snapshot as in this case. For the complete list of key names that might occur in the Protocol Files, see the table in "DB2 Advanced Copy Services (ACS) protocol file" in the DB2 Information Center for DB2 10.5. (http://www-01.ibm.com/support/docview.wss?uid=swg27023558)

The option DB2BACKUP_MODE can take the values ONLINE and OFFLINE and is used to reflect online or offline snapshots. The default is OFFLINE. DB2BACKUP_LOGS takes the values INCLUDE or EXCLUDE depending on whether include or exclude logs was specified in the DB2 backup command. The default value is INCLUDE.

Prepare

After this information has been written to the Protocol File, the Customer Script is invoked for the first time with the action prepare:

```
# ==========================================================
# db2ACSPrepare(): BEGIN [Mon Apr 22 05:00:23 2013]
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a prepare
   -c /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
   /repository 2ndoption
RC_PREPARE=0
# db2ACSPrepare(): END [Mon Apr 22 05:00:23 2013]
# ==========================================================
```

After that, the next call is the snapshot call. Before this call – if the snapshot is taken online - write operations of the database are suspended, that is, WRITE SUSPEND is set automatically on the database. Therefore, the prepare call is the right place to prepare file systems, check space requirements in the storage system and other things in advance before the snapshot is actually taken. The Sample Customer Script does nothing during that call.

Snapshot

Now that the preparation is done, DB2 can invoke the script to actually do the snapshot. Again, before the invocation, DB2 sets the database in WRITE SUSPEND mode. If exclude logs was specified, writing to the log files will continue (for further information on SET WRITE command take a look in the DB2 Information Center). If the Customer Script does not write anything to the Protocol File, the section for the db2ACSSnapshot would look like the following:

```
# ========================================================
# db2ACSSnapshot(): BEGIN [Mon Apr 22 05:00:23 2013]
OBJ_ID=0
ACTION=DB2ACS_ACTION_WRITE
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a snapshot
   -c /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
   /repository 2ndoption
RC_SNAPSHOT=0
# db2ACSSnapshot(): END [Mon Apr 22 05:00:25 2013]
# ========================================================
```

The Sample Customer Script packs two archives: one for the data paths and one for both log paths (if they exist and log files include was specified).

To get a snapshot of the data files, the Sample Customer Script takes all paths whose option names start with DATAPATH and packs them to this tar archive by performing the following steps:

1. Generating the file name for the data files
   ```
   file="${repository}${db_name}.0.${instance}.${dbpartnum}.${tim
   estamp}.001.tar"
   ```
   As you can see, the database name, the instance name, the partition number, and the timestamp of the snapshots are used for the name. The names are again oriented on the names of common DB2 backups.
2. Storing the file name in the Protocol File
   ```
   storeSetting "BACKUP_FILE" $file
   ```
3. Generating the command:
   ```
   cmd="awk -F= '/^DATAPATH/ { print \$2; }' $config | xargs tar
   -cf $file 2>/dev/null && echo 0 || echo 1"
   ```
4. Writing the command to the Protocol File:
   ```
   echo "# cmd: $cmd"
   ```
5. Running the command and reading the return code:
   ```
   RC=`eval $cmd`
   ```
6. Writing the return code to the Protocol File:
   ```
   echo "# backup tar created, rc=$RC"
   ```

Now, the script has to take care of the log files. First, it determines if log files have to be included by reading the value for the corresponding option:

```
getSetting "DB2BACKUP_LOGS"
includeLogs=$_setting
```

Depending on this result, actions are taken:

```
if [ $includeLogs = "INCLUDE" ]
then
   echo "# Logs to be included"
```

The name of the tar archive for the log files is generated the same way as the tar name for the tar file of the data paths and also stored in the Protocol File. The tar file for the log files is also packed the same way as the tar file for the data paths. This time the values for the options starting with LOGPATH are used in the following steps:

1. Generating the file name for the data files
   ```
   file="${repository}${db_name}.0.${instance}.${dbpartnum}.${tim
   estamp}.log.tar"
   ```
   The name is similar to the tar file for the data files but with the string "log" instead of "001".
2. Writing the file name in the Protocol File
   ```
   storeSetting "BACKUP_FILE" $file
   ```
3. Generating the command:
   ```
   cmd="awk -F= '/^LOGPATH/ { print \$2; }' $config | xargs tar -
   cf $logs 2>/dev/null && echo 0 || echo 1"
   ```
4. Writing the command in the Protocol File:
   ```
   echo "# cmd: $cmd"
   ```
5. Running the command and read the return code:
   ```
   RC=`eval $cmd`
   ```
6. Writing the return code to the Protocol File:
   ```
   echo "# tar for logs created, rc=$RC"
   ```

The following snippet from a Protocol File shows the complete contents of the snapshot call including all comments written by the Sample Customer Script:

```
# ==========================================================
# db2ACSSnapshot(): BEGIN [Mon Apr 22 05:00:23 2013]
OBJ_ID=0
ACTION=DB2ACS_ACTION_WRITE
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a snapshot
   -c /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
# /repository 2ndoption
BACKUP_FILE=/repository/JK1.0.db2jk1.0.20130422050024.001.tar
# cmd: awk -F= '/^DATAPATH/ { print $2; }'
   /repository/db2acs.JK1.0.db2jk1.1366621223.cfg |
   xargs tar -cf
   /repository/JK1.0.db2jk1.0.20130422050024.001.tar
   2>/dev/null && echo 0 || echo 1
# backup tar created, rc=0
# Logs to be included
BACKUP_LOGS=/repository/JK1.0.db2jk1.0.20130422050024.log.tar
# cmd: awk -F= '/^LOGPATH/ { print $2; }'
   /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
   | xargs tar -cf
   /repository/JK1.0.db2jk1.0.20130422050024.log.tar
   2>/dev/null && echo 0 || echo 1
# tar for logs created, rc=0
RC_SNAPSHOT=0
# db2ACSSnapshot(): END [Mon Apr 22 05:00:25 2013]
# ==========================================================
```

Immediately after the Customer Script has finished the snapshot action and the control was given back to DB2, write operations of the database are allowed again, that is, WRITE RESUME is automatically set.

Verify

The next call that follows is the verify call to check if the snapshot was taken successfully. To do so, the Sample Customer Script just checks if the needed tar files exist (test with –f) and if the size is greater than zero (test for –s), that is, if the tar file of the data files exists as follows:

```
getSetting "BACKUP_FILE"
file=$_setting
if [ -f "$file" -a -s "$file" ]
```

If include logs was specified, the Sample Customer Script also checks if the tar of the log files exists as follows:

```
getSetting "BACKUP_LOGS"
logs=$_setting
getSetting "DB2BACKUP_LOGS"
includeLogs=$_setting
if [ $includeLogs = "INCLUDE" ]
then
   if [ -f "$logs" -a -s "$logs" ]
```

If one of these criteria is not met, RC_ERROR is returned.

If everything worked correctly, the following output in the Protocol File might have been produced:

```
# ==========================================================
# db2ACSVerify(): BEGIN [Mon Apr 22 05:00:25 2013]
FIRST_ACTIVE_LOG_ID=1
FIRST_ACTIVE_LOG_CHAIN=0
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a verify
   -c /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
   /repository 2ndoption
# Backup '/repository/JK1.0.db2jk1.0.20130422050024.001.tar' exist
# Logs '/repository/JK1.0.db2jk1.0.20130422050024.log.tar' exist
RC_VERIFY=0
# db2ACSVerify(): END [Mon Apr 22 05:00:25 2013]
# ==========================================================
```

This snippet contains two new options that describe the log files that were in use during the snapshot (FIRST_ACTIVE_LOG_ID) and in which log chain they were written (FIRST_ACTIVE_LOG_CHAIN). Both options are needed in case of a restore and in particular during a rollforward. The snippet above also contains the success messages of the checks for the existence of the tar files in the verify call in the Sample Customer Script.

Note that depending on the result of the verify call, store metadata in the case of success and rollback in the case of failure is called.

Storemetadata

Let us assume the verify call was successful; then the next call will be storemetadata. This is the right place to save additionally required files that were produced during the snapshot.We recommend, for example, that you save the Protocol File to the backup infrastructure: there will be further information written to it but the restores will succeed with the current status.

```
# =========================================================
# db2ACSStoreMetaData(): BEGIN [Mon Apr 22 05:00:25 2013]
START_TIME=1366621224
METADATA_SIZE=12364
METADATA_CHECKSUM=20058929
METADATA=...
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a store_metadata
   -c /repository/db2acs.JK1.0.db2jk1.1366621223.cfg
   /repository 2ndoption
RC_STORE_METADATA=0
# db2ACSStoreMetaData(): END [Mon Apr 22 05:00:25 2013]
# =========================================================
```

You will recognize that the METADATA option takes the longest value. It consists of a memory block of DB2 that is encoded in Base64. Any change in this block will make the snapshot image unusable. For DB2 to be able to check the validity during the restore the METADATA_CHECKSUM is added. The METADATA_SIZE describes the size of this memory block.

The Sample Customer Script does nothing during this call.

Rollback

The verify call returned a failure during the snapshot action; the Customer Script is now called with the action rollback. During this call you should clean everything that might have been created. The Sample Customer Script extracts the required names for the tar files from the candidate Protocol File and deletes them.

```
doRollback() {
   getSetting "BACKUP_FILE"
   oldBackup=$_setting
   echo "# Delete old backup file : $oldBackup"
   rm $oldBackup

   getSetting "DB2BACKUP_LOGS"
   includeLogs=$_setting
   if [ $includeLogs = "INCLUDE" ]
   then
      getSetting "BACKUP_LOGS"
      oldLogs=$_setting
      echo "# Delete old backup file : $oldLogs"
      # Delete old logs file
      rm $oldLogs
   fi
}
```

<u>Terminating the snapshot</u>

After all calls for the snapshot operation are done, DB2 terminates the session and the operation. The return code of the complete operation (RC_OPERATION) is added to the Protocol File. If the value is *0*, the operation completed successfully, and the image can be used for a restore; return codes other than *0* mean that the image cannot be used for restores.

```
# =========================================================
# db2ACSEndOperation(): BEGIN [Mon Apr 22 05:00:25 2013]
RC_OPERATION=0
# db2ACSEndOperation(): END [Mon Apr 22 05:00:25 2013]
# =========================================================
# db2ACSTerminate(): BEGIN [Mon Apr 22 05:00:25 2013]
# db2ACSTerminate(): END [Mon Apr 22 05:00:25 2013]
# =========================================================
```

Restore

During restores the Customer Script is only called three times: for prepare, for restore, and, if the restore failed, for rollback. The snippets below are taken from a restore that failed since the Protocol Files of successful restores will be disregarded.

Note that during restore, query, and delete operations, two Protocol Files are opened: one that describes the current operation and a second one – temporarily open and read-only - that describes the snapshot image that currently is under consideration. In the following sections, the last one will be called candidate Protocol File.

<u>Prepare</u>

During the prepare phase, the Customer Script may provide additionally required Protocol Files in the Protocol File Repository.

Before calling the Customer Script, DB2 already provides some information about the database that has to be restored and the timestamp, as shown in the following snippet:

```
# =========================================================
# db2ACSBeginOperation(): BEGIN [Tue May 21 09:26:54 2013]
OPERATION=RESTORE
# db2ACSBeginOperation(): END [Tue May 21 09:26:54 2013]
# =========================================================
# db2ACSBeginQuery(): BEGIN [Tue May 21 09:26:54 2013]
QUERY_TYPE=SNAPSHOT
QUERY_DBPARTNUM=0
QUERY_DB=JK1
QUERY_INSTANCE=*
QUERY_HOST=*
QUERY_OWNER=*
QUERY_TIMESTAMP=20130521092650
```

```
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a prepare
   -c /repository/db2acs.JK1.0.db2jk1.1369142814.cfg
   /repository
RC_PREPARE=0
# db2ACSBeginQuery(): END [Tue May 21 09:26:54 2013]
# ============================================================
```

Querying

It is possible to specify only part of the exact timestamp to the restore command, for example, only the date. DB2 will filter by the specified date and, in the output, return the images that meet this partial timestamp, that is, all images that were taken on this day. The candidates are listed in recurring db2ACSGetNextObject calls like the following:

```
# ============================================================
# db2ACSGetNextObject(): BEGIN [Tue May 21 09:26:54 2013]
RESULT_0_FILE=/repository/db2acs.JK1.0.db2jk1.1369142810.cfg
# Timestamp 20130521092650
# db2ACSGetNextObject(): END [Tue May 21 09:26:54 2013]
# ============================================================
```

The candidate Protocol Files are enumerated according to these calls such that the next candidate would take the option RESULT_1_FILE.

After the last image was found, you will see an empty call to this function and afterwards the end of the query with the db2ACSEndQuery call.

```
# ============================================================
# db2ACSGetNextObject(): BEGIN [Tue May 21 09:26:54 2013]
# db2ACSGetNextObject(): END [Tue May 21 09:26:54 2013]
# ============================================================
# db2ACSEndQuery(): BEGIN [Tue May 21 09:26:54 2013]
# db2ACSEndQuery(): END [Tue May 21 09:26:54 2013]
# ============================================================
```

DB2 chooses the latest image of the candidates and restores it. If you asked, for example, for 20130521 and there are two images - the first one taken at 201305210900 and the second at 201305211000 - the last one is restored. If you want to restore a certain image you should specify the timestamp more exactly.

If the candidate was chosen, DB2 retrieves the metadata from the candidate Protocol File and decodes it, which can be seen in the following snippet:

```
# ============================================================
# db2ACSRetrieveMetaData(): BEGIN [Tue May 21 09:26:54 2013]
GET_META_OBJ_ID=0
METADATA_DECODED_SIZE=9272
METADATA_CHECKSUM=-1834173632
# db2ACSRetrieveMetaData(): END [Tue May 21 09:26:54 2013]
# ============================================================
```

These functions run without interaction with the Customer Script by DB2 alone.

<u>Restore</u>

During the restore call, the Sample Customer Script simply unpacks the data tar file and, if necessary, also the log tar file. DB2 provides the option OBJ_ID before the restore call is done. With the help of this number, the Protocol File for the backup image to be restored has to be retrieved as follows:

- Reading the OBJ_ID:
  ```
  getSetting "OBJ_ID"
  objectId=$_setting
  ```
- Build the option to be retrieved:
  ```
  key="RESULT_${objectId}_FILE"
  ```
- Reading the following option from the current Protocol File for the restore (the value for this option will be the name of the Protocol File for the backup image to be restored):
  ```
  getSetting $key
  oldConfig=$_setting
  ```

The following steps can now be performed:

- Reading the name of the tar file that was used for the data files from the candidate Protocol File:
  ```
  getSetting "BACKUP_FILE" "" $oldConfig
  oldBackup=$_setting
  ```
- Generating the command to unpack the tar file:
  ```
  cmd="tar -xf $oldBackup && echo 0 || echo 1"
  ```
- Writing the command to the Protocol File:
  ```
  echo "# cmd: $cmd"
  ```
- Running the command and tracking the return code:
  ```
  RC=`eval $cmd`
  echo "# tar extracted, rc=$RC"
  ```

During the restore, DB2 provides the option ACTION in the Protocol File. This option can have the following values: DB2ACS_ACTION_READ_BY_OBJECT or DB2ACS_ACTION_READ_BY_GROUP. The first one means that the complete database has to be restored, including the log files. The second value means that only the data files have to be restored. This is also handled by the Sample Customer Script taking the following steps:

- Reading the following option:
  ```
  getSetting "ACTION"
  readAction=$_setting
  ```
- Checking the following value (additionally checking if the unpacking of the data files completed successfully):
  ```
  if [ $readAction = "DB2ACS_ACTION_READ_BY_OBJECT" -a $RC -eq 0 ]
  then
  ```
- Running the corresponding commands to restore the log files like for restoring the data files:
  ```
  getSetting "BACKUP_LOGS" "" $oldConfig
  oldLogs=$_setting
  ```

```
      cmd="tar -xf $oldLogs && echo 0 || echo 1"
      echo "# cmd: $cmd"
      RC=`eval $cmd`
```

The following snippet reflects these actions in one of the Protocol Files for the restore:

```
# ==========================================================
# db2ACSSnapshot(): BEGIN [Tue May 21 09:26:54 2013]
OBJ_ID=0
ACTION=DB2ACS_ACTION_READ_BY_GROUP
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a restore
   -c /repository/db2acs.JK1.0.db2jk1.1369142814.cfg
   /repository
# cmd: tar -xf /repository/JK1.0.db2jk1.0.20130521092650.001.tar &&
echo 0 || echo 1
# tar extracted, rc=0
RC_RESTORE=4
# db2ACSSnapshot(): END [Tue May 21 09:26:54 2013]
# ==========================================================
```

Rollback

If the restore call failed as it did in the following example, the Customer Script is again invoked by DB2 with the action rollback. The following snippet shows the call for the rollback action. The Sample Customer Script also does nothing during this call.

```
# ==========================================================
# db2ACSEndOperation(): BEGIN [Tue May 21 09:26:54 2013]
RC_OPERATION=1
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a rollback
   -c /repository/db2acs.JK1.0.db2jk1.1369142814.cfg /repository
RC_ROLLBACK=0
# db2ACSEndOperation(): END [Tue May 21 09:26:54 2013]
# ==========================================================
```

The operation is finally terminated by DB2.

```
# ==========================================================
# db2ACSTerminate(): BEGIN [Tue May 21 09:26:54 2013]
# db2ACSTerminate(): END [Tue May 21 09:26:54 2013]
# ==========================================================
```

Query

You can use db2acsutil to delete images from a certain Protocol File Repository or to query a certain Protocol File Repository. The following syntax is now supported:

```
db2acsutil script "/home/db2jk1/sqllib/samples/BARVendor/libacssc.sh"
options "/repository" query status
```

The rules for the Protocol File Repository also apply to the db2acsutil command:

- If options are provided, the first option is always used as the Protocol File Repository.
- If no options are provided, the directory of the Customer Script is used.

All possible flags for db2acsutil are also supported. Therefore, you can filter by database, instance, host name, database partition number, and, of course, timestamp like in the following command:

```
db2acsutil script "/home/db2jk1/sqllib/samples/BARVendor/libacssc.sh"
options "/repository"
query status
taken at 20130521
database JK1
instance db2jk1
dbpartitionnum 0
show detail
```

A possible result can look as follows:

```
                       Instance : db2jk1
                       Database : JK1
                      Partition : 0
                Image timestamp : 20130522051311
                           Host : hal9000
                          Owner :
                    DB2 Version : 10.5.0
                  Creation time : Wed May 22 05:13:11 2013
  First active log (chain:file) : 2:13
                 Metadata bytes : 12364
                 Progress state : Successful
                Usability state : Unknown
                Bytes completed : 0
                    Bytes total : 0
```

The output specifies on which database in which instance the snapshot was taken and when it was taken. The progress state shows ”Successful” if the option OPERATION has the value 0 in the Protocol File, “Failed” if the option OPERATION has any other value than 0, and “In Progress” if it is not yet present in the Protocol File. The usability state always shows “Unknown” for images taken by the Scripted Interface because DB2 does not know if it is really usable.

The prepare action is the only action the Customer Script is invoked with during the query operation. The information given before that call is essentially the same as during the other prepare calls, as can be seen in the following snippet:

```
# ====================================================================
# db2ACSBeginQuery(): BEGIN [Wed May 22 08:17:40 2013]
QUERY_TYPE=ALL
QUERY_DBPARTNUM=-1
QUERY_DB=*
QUERY_INSTANCE=*
QUERY_HOST=*
QUERY_OWNER=*
```

```
QUERY_TIMESTAMP=*
OPERATION=QUERY
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh -a prepare -c
/home/db2jk1/repository/db2acs.0.1369225060.cfg /home/db2jk1/repository
2ndoption
RC_PREPARE=0
# db2ACSBeginQuery(): END [Wed May 22 08:17:40 2013]
# ====================================================================
```

The prepare call may provide Protocol Files in the Protocol File Repository that are, for example, restored from the backup infrastructure or any other actions. The prepare call is not implemented in the Sample Customer Script. The section db2ACSBeginQuery already shows criteria by which the candidates are filtered.

Afterwards, DB2 loops over all available Protocol Files and filters them by the given criteria, including operation SNAPSHOT. The candidates that DB2 considers are shown in the Protocol File as in the following example:

```
# ====================================================================
# db2ACSGetNextObject(): BEGIN [Wed May 22 05:13:28 2013]
RESULT_2_FILE=/home/db2jk1/repository/db2acs.JK1.0.db2jk1.1369214000.cf
g
# Timestamp 20130522051322
# db2ACSGetNextObject(): END [Wed May 22 05:13:28 2013]
# ====================================================================
```

The output will be given for all candidates that meet the criteria.

The operation will finally be closed, which is shown in the following snippet:

```
# ====================================================================
# db2ACSGetNextObject(): BEGIN [Wed May 22 08:17:40 2013]
# db2ACSGetNextObject(): END [Wed May 22 08:17:40 2013]
# ====================================================================
# db2ACSEndQuery(): BEGIN [Wed May 22 08:17:40 2013]
# db2ACSEndQuery(): END [Wed May 22 08:17:40 2013]
# ====================================================================
# db2ACSTerminate(): BEGIN [Wed May 22 08:17:40 2013]
# db2ACSTerminate(): END [Wed May 22 08:17:40 2013]
# ====================================================================
```

Delete

It is also possible to use db2acsutil to delete images taken with the Scripted Interface, e.g. with the following command:

```
db2acsutil script "$HOME/sqllib/samples/BARVendor/libacssc.sh" options
"/repository" query status
```

Again, all additional filter options apply.

With this command, the first action that is used to call the script is again a prepare call that should take similar actions like during restore or query operations:

```
# ======================================================================
# db2ACSBeginOperation(): BEGIN [Wed May 22 05:13:28 2013]
OPERATION=DELETE
# db2ACSBeginOperation(): END [Wed May 22 05:13:28 2013]
# ======================================================================
# db2ACSBeginQuery(): BEGIN [Wed May 22 05:13:28 2013]
QUERY_TYPE=ALL
QUERY_DBPARTNUM=-1
QUERY_DB=SAMPLE
QUERY_INSTANCE=*
QUERY_HOST=*
QUERY_OWNER=*
QUERY_TIMESTAMP=*
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a prepare
   -c /repository/db2acs.0.1369214008.cfg
   /repository
RC_PREPARE=0
# db2ACSBeginQuery(): END [Wed May 22 05:13:28 2013]
# ======================================================================
```

After the prepare call, DB2 starts again a query over all available Protocol Files and lists the candidates in the current Protocol File:

```
# ======================================================================
# db2ACSGetNextObject(): BEGIN [Wed May 22 05:13:28 2013]
RESULT_0_FILE=/repository/db2acs.JK1.0.db2jk1.1369213989.cfg
# Timestamp 20130522051311
# db2ACSGetNextObject(): END [Wed May 22 05:13:28 2013]
# ======================================================================
```

Afterwards, DB2 calls db2ACSDelete for every image to be deleted what in turn invokes the Customer Script with the delete action:

```
# ======================================================================
# db2ACSDelete(): BEGIN [Wed May 22 05:13:29 2013]
DELETE_OBJ_ID=0
# cmd: /home/db2jk1/sqllib/samples/BARVendor/libacssc.sh
   -a delete
   -o 0
   -t 20130522051311
   -c /repository/db2acs.0.1369214008.cfg
   /repository
# Delete old backup file and logs:
/repository/SAMPLE.0.db2jk1.0.20130522051311.001.tar
# Delete old backup file :
/repository/SAMPLE.0.db2jk1.0.20130522051311.log.tar
RC_DELETE=4
# db2ACSDelete(): END [Wed May 22 05:13:29 2013]
# ======================================================================
```

To make the handling easier, this call uses two other options: The flag *–o* that indicates the object ID of the image to be deleted in this action and additionally the timestamp of this image that is provided by the flag *–t*. The Sample Customer Script deletes the tar files by performing the following steps:

- Building the option to read the appropriate Protocol File:
  `key="RESULT_${objectId}_FILE"`

- Reading the option and parsing the value:
  ```
  getSetting $key
  oldConfig=$_setting
  ```
- Reading the name of the data tar file from this Protocol File:
  ```
  getSetting "BACKUP_FILE" "" $oldConfig
  oldBackup=$_setting
  ```
- Deleting the data tar file:
  ```
  rm $oldBackup
  ```
- Reading the option DB2BACKUP_LOGS to check if logs were included:
  ```
  getSetting "DB2BACKUP_LOGS" "" $oldConfig
  includeLogs=$_setting
  ```
- If the log files were included:
  ```
  if [ $includeLogs = "INCLUDE" ]
  then
  ```
- Reading the name of the tar file used for the logs
  ```
  getSetting "BACKUP_LOGS" "" $oldConfig
  oldLogs=$_setting
  ```
- Removing the log files:
  ```
  rm $oldLogs
  ```

After each successful invocation of the delete action to the Customer Script, the corresponding Protocol File is deleted by DB2.

The delete operation will be terminated, which is also shown in the Protocol Files as follows:

```
# =====================================================================
# db2ACSEndQuery(): BEGIN [Wed May 22 05:13:29 2013]
# db2ACSEndQuery(): END [Wed May 22 05:13:29 2013]
# =====================================================================
# db2ACSEndOperation(): BEGIN [Wed May 22 05:13:29 2013]
RC_OPERATION=0
# db2ACSEndOperation(): END [Wed May 22 05:13:29 2013]
# =====================================================================
# db2ACSTerminate(): BEGIN [Wed May 22 05:13:29 2013]
# db2ACSTerminate(): END [Wed May 22 05:13:29 2013]
# =====================================================================
```

Conclusion

This article introduced the new Scripted Interface for DB2 Advanced Copy Services. It explained the Protocol Files, the Customer Scripts and every operation in detail. The next parts of this series will take a look at more real-world examples.