developerWorks article template using OpenDocument

Important: Please ensure that your input in this information form is inside the fields. To enter data in a very narrow field , press ctrl-shift-F9 and then use the **Next** button to move through fields.

Type of Submission:
Title: The Scripted Interface for DB2 Advanced Copy Services
Subtitle: Part 3 – Implementation of the Scripted Interface for DB2 ACS Using IBM GPFS

Keywords: DB2 ACS, scripted interface, GPFS, snapshot

Given: Martin
Family: Jungfer

Job Title: software engineer
Email: jungfer@de.ibm.com
Bio: See below.
Company: IBM Deutschland Research and Development GmbH
Photo filename: jungfer.jpg

Job Title: Staff Development Engineer
Email: joern.klauke@de.ibm.com
Bio: See below.
Company: IBM Deutschland Research and Development GmbH
Photo filename: Foto_klein.jpg

Abstract: See below.

Bio Martin: Martin Jungfer works at IBM Information Management as a software engineer in the IBM DB2 for Linux, Unix and Windows for SAP development team in Boeblingen, Germany. He has more than ten years of experience with SAP on IBM DB2 for Linux, Unix and Windows. He holds a degree in technical computer science from the University for Applied Science in Albstadt, Germany.

Bio Joern: Joern Klauke works as 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 LUW assisting customers with best practices, problem analysis and troubleshooting. He holds a Diploma degree in Computer Science from the Martin-Luther-University of Halle (Germany).

Abstract: The IBM® DB2® Advanced Copy Services (DB2 ACS) support taking snapshots for backup purposes in DB2 for Linux®, Unix® and Windows® databases. You can use the DB2 ACS API either through libraries implemented by your storage hardware vendors (wheras until now, only some do) or you can implement this API yourself which however, involve a high effort. This changes with IBM DB2 10.5.

As of IBM DB2 10.5, a new feature called Scripted Interface for DB2 Advanced Copy Services is introduced. 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. The featur is supported on all UNIX and Linux platforms DB2 is certified on.

In this series we will provide an introduction to the Scripted Interface feature and present a real life example. This is the third part of the series and demonstrates the usage of the Scripted Interface together with IBM General Parallel Filesystem (IBM GPFS). The DB2 database in this example is used by an SAP® NetWeaver® system. IBM GPFS is configured as recommended by SAP in their installation guide for DB2 for Linux, UNIX and Windows with the pureScale feature [1].

Introduction to IBM GPFS

The IBM General Parallel File System (GPFS) file system is a high-performance shared disk file management solution that provides fast, reliable access to a common set of file data from two computers up to hundreds of systems. A GPFS file system integrates into your environment by bringing together mixed server and storage components to provide a common view of enterprise file data [2].

One key capability of the IBM GPFS is its functionality to create snapshots of file systems or file sets at a single point in time. This will be used for database backup and restores with the Scripted Interface. Other key capabilities such as its high reliability, high performance, high availability and flexibility are not in the focus of this series.

Hardware and Software Details

The following describes the hardware and software used for the implementation. Three LPARs running on AIX 6.1 are connected to a SAN storage subsystem. The GPFS file systems are located on the SAN storage subsystem and are shared among all 3 LPARs. On each LPAR, the DB2 software and instance directories are installed on local file systems. Shared GPFS file systems are used for DB2 tablespace containers, database directories, and transaction log directories.

Figure 1: Hardware and Software Details

IBM GPFS Configuration

This chapter briefly describes the IBM GPFS configuration used in this document. It shows the differences between GPFS file systems and regular AIX journaled file systems used in a DB2 installation relevant for GPFS snapshots. For more detailed information about IBM GPFS and its components, refer to the official IBM product documentation [3] and the SAP Installation Guide [1].

GPFS Network Shard Disks

Starting from bottom up, we use logical drives provided to the AIX host by a storage subsystem. On AIX, the logical drives are available as hdisks and can be listed with the command `lspv`. In the listing below, hdisk0 is used for the AIX rootvg. All other hdisks are allocated to IBM GPFS.

```
# lspv
hdisk0          00c5cc44744335cb                    rootvg
hdisk3          00c5cc4468e04c1f                    gpfs6nsd
hdisk4          00c5cc4468e29d49                    gpfs7nsd
hdisk5          00c5cc4468e5a943                    gpfs8nsd
hdisk6          00c5cc4468e6b502                    gpfs9nsd
hdisk7          00c5cc4468e99f8c                    gpfs10nsd
hdisk8          00c5cc4468eaaf55                    gpfs11nsd
hdisk9          00c5cc4468eca8ad                    gpfs3nsd
hdisk10         00c5cc4468ed9435                    gpfs13nsd
hdisk11         00c5cc4468ee4925                    gpfs2nsd
hdisk12         00c5cc4468eef3a3                    gpfs12nsd
hdisk1          00c5cc4468f1322e                    gpfs1nsd
hdisk15         00c5cc146e295724                    gpfs4nsd
hdisk16         00c5cc146e293b37                    gpfs5nsd
```

During initial IBM GPFS configuration, hdisks are mapped to so-called Network Shared Disks (NSDs) by using the GPFS command `mmcrnsd`. This mapping is reflected in the `lspv` output. The rightmost output column shows the GPFS NSD name in addition to the AIX volume group. For example, the NSD gpfs5nsd is mapped to AIX hdisk16.

GPFS File Systems

Next, NSDs are used to make up a GPFS file system. The existing configuration can be displayed with the `mmlsnsd` command:

```
# /usr/lpp/mmfs/bin/mmlsnsd

 File system   Disk name    NSD servers
---------------------------------------------------------------------------
 database_dir  gpfs2nsd     (directly attached)
 db2dump_dir   gpfs12nsd    (directly attached)
 db2fs1        gpfs1nsd     (directly attached)
 log_dir       gpfs3nsd     (directly attached)
 log_dir2      gpfs13nsd    (directly attached)
 sapdata1      gpfs4nsd     (directly attached)
 sapdata2      gpfs5nsd     (directly attached)
 sapdata3      gpfs6nsd     (directly attached)
 sapdata4      gpfs7nsd     (directly attached)
```

```
sapdata5      gpfs8nsd     (directly attached)
sapdata6      gpfs9nsd     (directly attached)
sapdata7      gpfs10nsd    (directly attached)
sapdata8      gpfs11nsd    (directly attached)
```

The output lists the GPFS file systems in the first column and the NSD in the second column. The third column states that the NSDs are directly attached to SAN disks without the use of NSD server. For example, the GPFS file system sapdata8 is located only on NSD gpfs11nsd. In terms of an AIX journaled file system, the NSD would be the logical volume. The creation of GPFS file systems is not in scope of this document. Please refer to official IBM GPFS documentation [3] and the SAP installation guide [1].

GPFS Nodes
Now with several GPFS file systems available, they can be mounted on multiple GPFS nodes in parallel. IBM GPFS controls concurrent access to the file systems and ensures consistency. The configuration can be displayed with the `mmlsmount` command for all or for only one file system, e.g., sapdata8.

```
# /usr/lpp/mmfs/bin/mmlsmount sapdata8 -L

File system sapdata8 is mounted on 3 nodes:
  12.34.567.12    host2
  12.34.567.34    host3
```

The output shows that the GPFS file system sapdata8 is mounted on 2 GPFS nodes, host2 and host3. The GPFS nodes provide the file systems to AIX, and finally DB2 pureScale members 2 and 3 can use shared file systems.

IBM GPFS allows much more dedicated configurations to achieve maximum reliability, availability, and performance. However, in this test environment we have a direct one-to-one relationship between hdisk, NSD, and the GPFS file system. This is illustrated in Figure 2.

Figure 2: Relationship between hdisk, NSD, shared file system and DB2 member

The IBM DB2 database uses all of the above GPFS file systems for its database directories, tablespaces, and transaction log files. These file systems allow shared access for each GPFS node, i.e. DB2 pureScale member. This is the basis so that DB2 backup and restore commands can be run from any member of the IBM DB2 pureScale environment.

In a standard SAP installation on DB2 using the pureScale feature in addition to the shared file systems, local file systems exist on each DB2 member, for example, the home directory of the instance owner and the DB2 software installation directory. SAP application-specific file systems are not in the scope of this document.

Table 1 gives an overview of the local and shared GPFS file systems used in the test environment. Note that the GPFS file systems and their mount points can have different names, e.g. GPFS file system db2dump_dir is mounted on mount point /db2/BWP/db2dump.

Remark:
In this test environment, multiple file systems for sapdata are shown (sapdata1...8). However, SAP recommends to use only one GPFS sapdata file system in DB2 pureScale installations [1].

file system mount point	file system type	GPFS file system name
/db2	local	-
/db2/instance_shared	shared GPFS	db2fs1
/db2/db2bwp	local	-
/db2/db2bwp/db2_software	local	-
/db2/db2bwp/sqllib	local	-
/db2/BWP	shared GPFS	database_dir
/db2/BWP/sapdata1...8	shared GPFS	sapdata1..8
/db2/BWP/log_dir	shared GPFS	log_dir
/db2/BWP/log_dir2	shared GPFS	log_dir2
/db2/BWP/db2dump	shared GPFS	db2dump_dir

Table 1: shared and local file systems used in test environment

Used GPFS Commands for DB2 Backup and Restore

The following table lists the GPFS commands that are used to perform the backup and restore work in the customer script. On AIX, the default directory of the commands is /usr/lpp/mmfs/bin.

Command	Usage
mmcrsnapshot	Creates a snapshot of a file system at a single point in time. The parameters are the file system name and the name of the snapshot to be created.
mmlssnapshot	Displays GPFS snapshot information for the specified file system. The command is used to verify that the snapshot state is valid.
mmdelsnapshot	The command is used to delete snapshots in case a DB2 backup failed or a DB2 backup is not needed any longer.
mmrestorefs	The command is used within the DB2 restore database processes to restore a file system from a snapshot.
mmumount	Unmounts a GPFS file system. It is used in the process of DB2 restore to unmount GPFS file systems before restoring the file system.
mmmount	Mounts a GPFS file system. It is used to mount GPFS file systems after a DB2 restore process.

Table 2: GPFS commands used in customer script

GPFS snapshots are software-based copies of file system data blocks. They do not use specific hardware functions of the storage subsystem. Therefore, the processing speed of the file system snapshots and restores is obviously slower compared to hardware-based snapshots performed by an IBM storage subsystem.

The GPFS snapshots are not complete copies of file systems. They keep the original of a data block, which would be changed or deleted after the snapshot was taken. They can be used to recover the file systems to the point in time when the snapshot was taken. The original file system must be intact. Therefore, the snapshots cannot protect against media failures. Please refer to official IBM GPFS documentation [3], chapter "Recoverability considerations" for details on this topic. The snapshots will be placed in the base directory of each GPFS file system under the name .snapshot. For example, the snapshots taken for the file system sapdata1, which is mounted on /db2/BWP/sapdata1, are located under subdirectory /db2/BWP_sapdata1/.snapshots/.

Example:

```
host3:db2bwp 27> pwd
/db2/BWP_sapdata1/.snapshots/snap_DATA_20130716103115/db2bwp/NODE0000..
.
```

Privileges for Running GPFS Commands

The customer script that is invoked by the Scripted Interface runs with the privileges of the instance owner. However, almost all of the GPFS commands listed above require root authority. For this part of the series, the privileges to run the GPFS commands were granted using sudo. The GPFS commands were added to the sudo configuration with the use of the command `visudo` with lines like the following:

```
db2bwp ALL=NOPASSWD: /usr/lpp/mmfs/bin/mmcrsnapshot
db2bwp ALL=NOPASSWD: /usr/lpp/mmfs/bin/mmlssnapshot
db2bwp ALL=NOPASSWD: /usr/lpp/mmfs/bin/mmdelsnapshot
db2bwp ALL=NOPASSWD: /usr/lpp/mmfs/bin/mmumount
db2bwp ALL=NOPASSWD: /usr/lpp/mmfs/bin/mmmount
db2bwp ALL=NOPASSWD: /usr/lpp/mmfs/bin/mmrestorefs
```

For other platforms, there might be other solutions, e.g., adding the instance owner to groups that are allowed to run the appropriate commands. We discourage you to use scripts with the setuid bit set.

Implementation of GPFS Snapshots for Scripted Interface

This chapter describes the implementation of the customer script using the snapshot capability of IBM GPFS. This includes all processing steps in the script from the point where it is called from DB2 via the ACS API. DB2 ACS API and the general use of the Scripted Interface functionality is documented in the official IBM documentation of DB2 10.5 [1] and in the previous parts of this developerWorks series.

The script coding is kept simple on purpose. Several lines could be saved by combining commands like awk, grep, and output evaluation into one line. However, this would make it harder to read for users who are not used to korn shell syntax.

General Environment Variables

The script calls several GPFS commands to perform the file system snapshot and restores.
The commands are defined by variables in the beginning of the script as shown in the following excerpt. These variables might need to be adapted to the environment.

The line numbers are only valid in this context. The actual line numbers in the script are different.

```
+19  # GPFS path and exes
+20  GPFSINST=/usr/lpp/mmfs
+21  GPFSBIN=$GPFSINST/bin
+22  GPFSSNAP=$GPFSBIN/mmcrsnapshot
+23  GPFSVFY=$GPFSBIN/mmlssnapshot
+24  GPFSDEL=$GPFSBIN/mmdelsnapshot
+25  GPFSREST=$GPFSBIN/mmrestorefs
+26  GPFSMOUNT=$GPFSBIN/mmmount
+27  GPFSUMOUNT=$GPFSBIN/mmumount
```

For example, the GPFS command to create a snapshot `mmcrsnapshot` is defined in line 22. Within the script, the variable $GPFSSNAP is used to reference the command.

The script writes only the minimal required information into the protocol file, such as the names of the GPFS snapshots. Useful information for error analysis of the customer script is written into a separate log file. The location and name are also defined in the script as follows:

```
+29  # Log file
+30  LOGPREFIX="gpfs_snap"
+31  LOGPOSTFIX="BWP"
+32  LOG=/tmp/${LOGPREFIX}_${LOGPOSTFIX}.log
```

These variables should also be adapted to the environment.

The script uses temporary files to save information needed during processing. The file names are also defined in the script and normally need not be changed. For example, the file defined by $FILESYSTEMS (line 42) will store the file system names to be snapped.

It will be read line by line and the GPFS snapshot command will be executed. In this way, large lists of file systems can be handled. See the following lines:

```
+38  # Tempfile names
+39  TMPDIR=/tmp
+40  TMP=${TMPDIR}/${LOGPREFIX}_${LOGPOSTFIX}.tmp
+41  TMP_=${TMPDIR}/${LOGPREFIX}_${LOGPOSTFIX}.tmp_
+42  file systems=${TMPDIR}/${LOGPREFIX}_${LOGPOSTFIX}_fs.tmp
```

During a DB2 backup using the Scripted Interface, it is possible to create a safe copy of the protocol file, which is implemented in this script. The path for the safe copy is also defined in the script. This variable might need to be adapted to the environment. See the following lines:

```
+34  # Protocol Backup directory
+35  PROT_BKP_DIR="/db2/db2bwp/scriptACS/prot_bkp/"
```

Data Read From Protocol File

The following lines are read from the protocol file when the script is called by the DB2 ACS API. Depending on the action specified by the -a option, some or all of them are evaluated. The lines are not changed by the customer script:

- Lines starting with the keyword DATAPATH_
- Lines starting with the keyword LOGPATH_
- Lines with the keyword DB2BACKUP_MODE, DB2BACKUP_LOGS
- Lines starting with the prefix RC_
- Lines with the keyword RESULT_x_FILE, where x is a variable number
- Line with the keyword OBJ_ID

Depending on how many lines are expected, they are read via the script function `getSetting` for a single line or with the UNIX `awk` command for multiple lines. These keywords are defined by the DB2 ACS API. Please refer to IBM documentation [1] for the detailed description of each keyword.

Data Written to the Protocol File

The following lines are written into the protocol file when the script is called by DB2 ACS API and the action specified by the -a option is “snapshot”. The lines are then inserted by the customer script with the script function `storeSetting`.

- Lines starting with the keyword USER_GPFSSNAP

These lines are used to save the information which GPFS snapshots relate to which DB2 backup image. This is the only place where this relation is stored. Therefore, it is very important to keep this file in a safe place.

The following is an example protocol file after a successful GPFS snapshot:

```
# db2ACSSnapshot(): BEGIN [Tue Jul 16 10:31:14 2013]
OBJ_ID=0
ACTION=DB2ACS_ACTION_WRITE
USER_GPFSSNAP=database_dir snap_DATA_20130716103114
USER_GPFSSNAP=sapdata1 snap_DATA_20130716103115
USER_GPFSSNAP=sapdata2 snap_DATA_20130716103117
USER_GPFSSNAP=sapdata3 snap_DATA_20130716103119
USER_GPFSSNAP=sapdata4 snap_DATA_20130716103120
USER_GPFSSNAP=sapdata5 snap_DATA_20130716103122
USER_GPFSSNAP=sapdata6 snap_DATA_20130716103123
USER_GPFSSNAP=sapdata7 snap_DATA_20130716103125
USER_GPFSSNAP=sapdata8 snap_DATA_20130716103127
# cmd: /db2/db2bwp/gpfs_sample_v07.sh -a snapshot -c
/db2/db2bwp/scriptACS/repository/db2acs.BWP.0.db2bwp.1373963472.cfg
/db2/db2bwp/scriptACS/repository
RC_SNAPSHOT=0
# db2ACSSnapshot(): END [Tue Jul 16 10:31:28 2013]
#
================================================================================
=====
```

The keyword “USER_GPFSSNAP” is also defined centrally in the beginning of the script by a variable. This variable might need to be adapted to the environment.

Action Prepare

This chapter describes what the script executes when it is called with the -a prepare parameter by the DB2 ACS API. The main script uses the same coding as already explained in part 1 of the series. The function `doPrepare` handles preparation work for all operations, that are, snapshot, restore and delete.

In case the preparation is to be done for a snapshot (i.e. DB2 database backup), the following steps are performed:

- Check if the GPFS commands required for the snapshot exist (function prepare_snapshot_command, line 610 below). If one of the commands does not exist, the action prepare is terminated with an error, causing the DB2 backup command to fail.
- Generate the list of GPFS file systems to be snapped (function get_used_file systems_for_backup, line 614 below). The list is generated according to the following logic:
    - First all lines starting with "DATAPATH_" are read from the protocol file.
    - If database log files should be included in the backup image, all lines starting with "LOGPATH_" are appended.
    - From this list of database paths and files, the associated file systems are retrieved with the AIX command `df -M`.
    - Typically, not every database path has its own file system. Therefore, this intermediate list most likely contains duplicates that must be removed. The final list is saved in a temporary file defined by variable $FILESYSTEMS. This file will then be used by the following call of the script with parameter -a snapshot.

In case the preparation is to be done for a restore (i.e. DB2 database restore), the following steps are performed:

- Check if the GPFS commands required for the restore exist (function prepare_restore_command, line 620 below). If one of the commands does not exist, the action prepare is terminated with an error, causing the DB2 restore command to fail.
- Next, the backup copies of the protocol files could be restored if needed. However, this is not implemented and should be done with care not to overwrite existing protocol files.
- The remaining logic is covered in the script function `doRestore` .

In case the preparation is to be done for a snapshot delete (i.e. db2acsutil delete backup image), the following steps are performed:

- Check if the GPFS commands required for the restore exist (function `prepare_restore_command`, line 630 below). If one of the commands does not exist, the action prepare is terminated with an error, causing the `db2acsutil` command to fail.

- The remaining logic is covered in the script function `doDelete`.

See the following excerpt (function doPrepare):

```
+576  ##################
+577  function doPrepare
…
583  {
…
+606     case $operation in
+607        "SNAPSHOT")
+608           # prepare for snapshot
+609           # check needed GPFS commands
+610           prepare_snapshot_command
+611           if_error "Error:  prepare_snapshot_command failed."
+612
+613           # get the file systems and store them in file
$FILESYSTEMS
+614           get_used_file systems_for_backup
+615
+616           ;;
+617        "RESTORE")
+618           # prepare for restore
+619           # check needed commands
+620           prepare_restore_command
+621           if_error "Error:  prepare_restore_command failed."
+622
+623           # copy  backup protocol files into place if the
repository is empty
+624           # get_used_file systems_for_restore, umount, restore,
mount
+625           # in doRestore function
+626           ;;
+627        "DELETE")
+628           # prepare for deletion of snapshot images
+629           # check needed commands
+630           prepare_delete_command
+631           ;;
+632         *)
+633           # default
+634           write_log "   Nothing specific to be prepared."
+635           ;;
+636     esac
```

Action Snapshot

This chapter describes what the script executes when it is called with the -a snapshot parameter by the DB2 ACS API. Function `doSnapshot` is used to handle the work for the action snapshot and uses the temporary file $FILESYSTEMS created by function `doPrepare` as input.

- The file $FILESYSTEMS is read line by line (lines 842 - 866). Each line consists of two fields:
  - The first field is the label “DATA” or “LOG”.
  - The second field is the name of the GPFS file system, for example:

    ```
    DATA /dev/database_dir
    LOG /dev/log_dir
    ```
- For each line a GPFS snapshot command is prepared in the variable $CMD (line 856).
- Two parameters for the GPFS command are also prepared and stored in the variables $GPFS_PARM1 and $GPFS_PARM2 (line 846, 853).
- $GPFS_PARM1 is the file system. $GPFS_PARM2 is the snapshot name consisting of a prefix (defined in the script), a type (DATA or LOG), and a timestamp.
- Next, the GPFS snapshot command is executed (line 858).
- In case of failure, the script is terminated with an error, causing the initial DB2 backup command to fail.
- In case of success, the name of the GPFS snapshot is written to the protocol file under the keyword $USER_GPFSSNAP with the function `storeSetting` (line 862).

See the following lines:

```
+830  function doSnapshot
+831  ###################
...
+834  {
...
+842     while read x
+843     do
...
+846        GPFS_PARM1=`echo $x | awk '{print $2}' | sed
's/^\/dev\///'`
+847
...
+851        TIMESTAMP=`date +'%Y%m%d%H%M%S'`
+852        TYPE=`echo $x | awk '{print $1"_"}'`
+853        GPFS_PARM2=${GPFS_SNAP_PREFIX}${TYPE}${TIMESTAMP}
+854
+855        write_log "$S GPFS snapshot ...."
+856        CMD="sudo $GPFSSNAP"
+857        write_log "   $CMD ${GPFS_PARM1} ${GPFS_PARM2}"
+858        eval $CMD ${GPFS_PARM1} ${GPFS_PARM2} >> $LOG 2>&1
+859
```

```
+860        if_error "Error: $CMD ${GPFS_PARM1} ${GPFS_PARM2} failed.
Exiting $0."
+861
+862        storeSetting "USER_GPFSSNAP=${GPFS_PARM1} ${GPFS_PARM2}"
+863
+864        write_log $D
+865
+866     done < $FILESYSTEMS
```

Action Restore

This chapter describes what the script executes when it is called with the -a restore parameter by the DB2 ACS API. Function `doRestore` is used to handle the work for the action restore. The temporary file $FILESYSTEMS is used again to store intermediate lists for GPFS file system restores.

- As a first step in this function, the correct protocol file for the restore is retrieved by reading the keyword "RESULT_x_FILE from the current protocol file (line 754 – 762).
- Next, the protocol file for the restore is used to generate a list of file systems to be restored (function get_used_file systems_for_restore, line 765).
    - First, all snapshot names are stored into a temporary file. This is done by reading all lines starting with "USER_GPFSSNAP" in the protocol file.
    - Afterwards, depending on the keyword "ACTION" in the restore protocol file, snapshot names containing the database log volumes are removed from the list.
    - The final list is stored in the temporary file $FILESYSTEMS again.
- Next, the file systems listed in the temp file $FILESYSTEMS are unmounted (line 768). In case of an error, the script is terminated with an error.
- Next, the file systems listed in the temp file $FILESYSTEMS are restored from their snapshots (function restore_file systems, line 771). All snapshots in the list are processed and return codes are collected. After the last one was processed, the function ends. In case of an error, the script is terminated with an error.

- Next, the file systems listed in the temp file $FILESYSTEMS are mounted again (line 775). In case of an error, the script is terminated.

See the following excerpt:

```
+741  function doRestore
...
+744  {
...
...
+755     getSetting "OBJ_ID"
+756     result_file_no=$_setting
+757
...
+760     key="RESULT_"${result_file_no}"_FILE"
+761     getSetting $key
```

```
+762     restoreConfig=$_setting
...
+765     get_used_file systems_for_restore
+766
+767     # unmount all
+768     umount_file systems
+769     if_error "Error: $CMD failed"
+770
+771     restore_file systems
+772     if_error "Error: $CMD failed"
+773
+774     # mount all
+775     mount_file systems
+776     if_error "Error: $CMD failed"
```

Action Verify

This chapter describes what the script is executing when it is called with the -a verify parameter by the DB2 ACS API. Function `doVerify` is used to handle the work for action verify.

- From the protocol file the lines starting with “USER_GPFSSNAP” are read and stored in a temporary file $TMP (lines 1048 - 1050). “USER_GPFSSNAP” is a user-defined keyword. The lines are written during the call of the script with -a snapshot action.
- This temporary file is read line by line in a while-do loop (lines 1053 – 1090). Each iteration calls the GPFS command mmlssnapshot with two parameters (lines 1056 - 1064):
  - $GPFS_PARM1 = the file system name (field 1 of the line)
  - $GPFS_PARM2 = the snapshot name (field 2 of the line)
- The output of the mmlssnapshot command is parsed for the field “Status”. If the field is not “Valid”, the variable TMP_RC is incremented by 1 (line 1086).
- After all snapshots were verified in the loop, the return code is set. If one of the snapshot images could not be verified, the return code of the script is set to $RC_VFY_ERROR (lines 1092 – 1101).

See the following excerpt:

```
+1029  ##################
+1030  function doVerify
...
+1048     CMD="awk -F= '/^USER_GPFSSNAP/ { print \$2 }' $config"
+1049     debug_info "data: using command: $CMD"
+1050     eval $CMD > $TMP
+1051
+1052     debug_info "data: reading from $TMP"
+1053     while read x
+1054     do
+1055       # prepare GPFS command parameters Device and Snapshot name
+1056       GPFS_PARM1=`echo $x | awk '{ print $1 }'`
+1057       GPFS_PARM2=`echo $x | awk '{ print $2 }'`
+1058
```

```
+1059       CMD="sudo $GPFSVFY ${GPFS_PARM1} -s ${GPFS_PARM2}"
+1060       write_log "   $CMD"
+1061
+1062       # store the output of the command in CMD_OUT
+1063       CMD_OUT=`eval $CMD 2>> $LOG`
+1064       RC=$?
+1065
+1066       if [[ $RC -eq 0 ]]
+1067       then
...
+1072
+1073          GPFS_SNAP_STATUS=`echo "$CMD_OUT" |  awk '{ if(NR==3)
print $3}'`
+1074          write_log "   ... returns: $GPFS_SNAP_STATUS."
+1075
+1076          if [[ $GPFS_SNAP_STATUS != "Valid" ]];
+1077          then
+1078             # increment counter, if command did not return
"Valid"
+1079             let "TMP_RC = TMP_RC + 1"
+1080             write_log "   ... Verification of snapshot failed."
+1081             write_log "   ... Status is not equal to \"Valid\"."
+1082          fi
+1083
+1084       else
+1085          # increment counter, if command failed in the loop
+1086          let "TMP_RC = TMP_RC + 1"
+1087          write_log "   ... Verification of snapshot failed."
+1088       fi
+1089
+1090     done < $TMP
+1091
+1092     # set the final return code
+1093
+1094     if [[ $TMP_RC -eq 0 ]]
+1095     then
+1096         write_log "   All snapshots successfully verified."
+1097         RC=0
+1098     else
+1099         write_log "   **** ERROR: at least one snapshot image \
                                      is not in state: valid.
rc=$RC_VFY_ERROR"
+1100         RC=$RC_VFY_ERROR
+1101     fi
```

Action StoreMetadata

This chapter describes what the script executes when it is called with the -a store_metadata parameter by the DB2 ACS API. Function `doStoreMetaData` is used to handle the work for this action. In case the snapshot (doSnapshot) and the verification (doVerify) are successful, this is the last call of the script before DB2 will internally process all the remaining backup work. Therefore, in function doStoreMetaData the following tasks areperformed:

- cleanup work for a successful GPFS snapshot (function cleanup_tempfiles, line 806)
- Create the safe copy of the protocol file in a separate directory (lines 808 – 819).

See the following excerpt:

```
+791  ##################
+792  function doStoreMetaData
+793  ##################
+794  # performs post processing after successful backup
...
+802     # Post Processing Tasks:
+803     # must be executed n both cases, if snapshot ok in Store
Metadata
+804     #                                  if  NOT in Rollback
+805     # cleanup
+806     cleanup_tempfiles
+807
+808     # save the protocol file
+809     CMD="cp $config $PROT_BKP_DIR"
+810     write_log "Starting saving the protocol file to another
directory"
+811     write_log "   $CMD"
+812     eval $CMD >> $LOG
+813     # give a warning instead of ERROR in this phase
+814
+815     if [[ $? -ne 0 ]]
+816        then
+817             # copy failed, print a warning in $LOG,
+818             write_log "   WARNING **** : protocol file could not
be saved"
+819     fi
```

Action Rollback

This chapter describes what the script executes when it is called with the -a rollback parameter by the DB2 ACS API. Function `doRollback` is used to handle rollback work for all actions, i. e. snapshot, verify, store_metadata, and restore.

In case the rollback has to be done for a failed snapshot or verify (i.e. DB2 database backup), the following steps are performed:

- From the protocol file the lines starting with "USER_GPFSSNAP" are read and stored in a temporary file $TMP (lines 926 - 928). "USER_GPFSSNAP" is a user-defined keyword. The lines are written during the call of the script with -a snapshot action.
- This temporary file is read line by line in a while-do loop (lines 934 – 951). Each iteration calls the GPFS command `mmdelsnapshot` with two parameters (lines 1056 - 1064):
  - $GPFS_PARM1 = the file system name (field 1 of the line)
  - $GPFS_PARM2 = the snapshot name (field 2 of the line)
- After all snapshots were processed in the loop, the return code is set. If one of the snapshot images could not be deleted, the return code of the script is set to $RC_RBCK_ERROR (lines 955 – 963).

In case the rollback has to be done for failed action store_metadata (i.e. DB2 database backup), the safe copy of the protocol file made in doStoreMetaData must be removed.

In case the rollback is to be done for failed action restore (i.e. DB2 database restore), we try to mount all file systems again to ensure the restore command can be repeated without having to manually mount the file systems.

In any case the function `cleanup_tempfiles` is called to remove the temporary files used (line 1019).

See the following excerpt:

```
+880  ##################
+881  function doRollback
...
+919     case $CMD_OUT in
+920        "RC_SNAPSHOT" | "RC_VERIFY")
...
+927           CMD="awk -F= '/^USER_GPFSSNAP/ { print \$2 }' $config"
+928           eval $CMD > $TMP
+929
+930           if [[ -s $TMP ]]
+931           then
…
+934             while read x
+935             do
+936                # prepare GPFS command parameters Device and
Snapshot name
+937                GPFS_PARM1=`echo $x | awk '{ print $1 }'`
```

```
+938                GPFS_PARM2=`echo $x | awk '{ print $2 }'`
+939
+940                CMD="sudo $GPFSDEL ${GPFS_PARM1} ${GPFS_PARM2}"
+941                write_log "   $CMD"
+942                eval $CMD 2>> $LOG
+943                RC=$?
+944
+945                if [[ $RC -ne 0 ]]
+946                then
+947                   # increment counter, if command failed in the
loop
+948                   let "TMP_RC = TMP_RC + 1"
+949                   write_log "   ... Deletion of snapshot failed."
+950                fi
+951             done < $TMP
...
+955             if [[ $TMP_RC -eq 0 ]]
+956             then
+957                 write_log "   All snapshots successfully deleted.
"
+958                 RC=0
+959             else
+960                 write_log "   **** ERROR: at least one snapshot
image could not\
+961                                be deleted. rc=$RC_RBCK_ERROR"
+962                 RC=$RC_RBCK_ERROR
+963             fi
+964
+965          else
+966             # No GPFS snapshot found for deletion
+967             write_log "   No GPFS snapshot to be deleted."
+968          fi
+969
+970          ;;
+971       "RC_STORE_METADATA")
…
+978          CMD="`echo $config  |  awk -F/ '{print \$NF }'`"
+979          write_log "   rm ${PROT_BKP_DIR}$CMD"
+980          eval "rm ${PROT_BKP_DIR}${CMD} 2>> $LOG"
+981
+982          if [[ $? -ne 0 ]]
+983          then
+984              write_log "   WARNING: File ${PROT_BKP_DIR}$CMD
could not be removed."
+985          fi
+986
+987          write_log $D
+988
+989          ;;
+990       "RC_RESTORE")
…
+996          mount_file systems
+997          if [[ $? -ne 0 ]]
+998          then
+999              write_log "   WARNING: file systems could not be
mounted "
+1000              RC=$RC_RBCK_ERROR
```

```
+1001           fi
+1002
+1003           write_log $D
+1004
+1005           debug_info "exit: case RC_RESTORE"
+1006
+1007           ;;
+1008        *)
+1009           # default, do nothing
+1010           write_log "Nothing specific to rollback for failed
step: $CMD_OUT"
+1011           ;;
+1012     esac
…
+1019     cleanup_tempfiles
```

Action Delete

This chapter describes what the script executes when it is called with the -a delete parameter by the DB2 ACS API. Function `doDelete` is used to handle the work for action delete.

- From the protocol file the lines starting with “USER_GPFSSNAP” are read and stored in a temporary file $TMP (lines 678 - 683). “USER_GPFSSNAP” is a user-defined keyword. The lines are written during the call of the script with -a snapshot action.
- This temporary file is read line by line in a while-do loop (lines 687 – 722). Each iteration calls the GPFS command `mmlsnapshot` with two parameters (lines 695 – 698) to check existence of the snapshot image:
  - $GPFS_PARM1 = the file system name (field 1 of the line)
  - $GPFS_PARM2 = the snapshot name (field 2 of the line)
- If the snapshot image does not exist, ignore any errors reported by the succeeding `mmdelsnapshot` command ($IGNORE_RC is set to 1, line 702). This behavior allows DB2 backups, for which GPFS snapshots were already deleted manually, to be removed.
- The GPFS command `mmdelsnapshot` is executed with the same two parameters as above. If the return code is not equal to 0 and $IGNORE_RC is equal 0 , errors of `mmdelsnapshot` are evaluated (i.e. increment $TMP_RC by 1 (line 711 - 718)) .
- After all snapshots were deleted in the loop, the return code is set. If one of the existing snapshot images could not be deleted, the return code is set to RC_DEL_ERROR. (lines 724 - 731)

See the following excerpt:

```
  +649  ##################
  +650  function doDelete
  +651  ##################
…
```

```
  +676     # for all lines starting with USER_GPFSSNAP, get the GPFS
device name
  +677     # and the snapshot name
  +678     CMD="awk -F= '/^USER_GPFSSNAP/ {print \$2\" \"\$3}'
$deleteConfig"
  +679
  +680     write_log $S "Retrieving GPFS snapshots from protocol
file..."
  +681     write_log "   writing them to $TMP"
  +682     debug_info "data: executing $CMD > $TMP"
  +683     eval $CMD > $TMP
  +684     if_error "Error: $CMD failed"
  +685
  +686     debug_info "data: reading from $TMP"
  +687     while read x
  +688     do
  +689        # set parm1 and parm2 for the GPFS commands
  +690        GPFS_PARM1=`echo $x | awk '{print $1}'`
  +691        GPFS_PARM2=`echo $x | awk '{print $2}'`
  +692
  +693        # if snap image exists exists evaluate RC (normal delete)
  +694        # if not ignore RC (cleanup delete)
  +695        CMD="sudo $GPFSVFY"
  +696        write_log "   $CMD ${GPFS_PARM1} -s ${GPFS_PARM2}"
  +697        eval $CMD ${GPFS_PARM1} -s ${GPFS_PARM2} >> $LOG 2>&1
  +698        RC=$?
  +699        if [[ $RC -ne 0 ]]
  +700        then
  +701          write_log "   Snap does not exist. Ignoring the
following $GPFSDEL error."
  +702          IGNORE_RC=1
  +703        fi
  +704
  +705        CMD="sudo $GPFSDEL"
  +706        write_log "   $CMD ${GPFS_PARM1} ${GPFS_PARM2}"
  +707        eval $CMD ${GPFS_PARM1} ${GPFS_PARM2} >> $LOG 2>&1
  +708        RC=$?
  +709
  +710        debug_info "data: RC: $RC and IGNORE_RC:$IGNORE_RC"
  +711        if [[ $RC -ne 0 ]] && [[ $IGNORE_RC -eq 0  ]]
  +712        then
  +713             # increment counter, if command did not return 0
  +714             let "TMP_RC = TMP_RC + 1"
  +715             debug_info "data: normal delete, report errors."
  +716             write_log "   WARNING **** : Can't delete this
snapshot:"
  +717             write_log "                ${GPFS_PARM1}
${GPFS_PARM2}"
  +718        fi
  +719
  +720        # reset the variable for next iteration
  +721        IGNORE_RC=0
  +722     done < $TMP
  +723
  +724     if [[ $TMP_RC -eq 0 ]]
  +725     then
  +726         write_log "   All snapshots deleted. Setting RC: 0."
```

```
  +727         RC=0
  +728     else
  +729         write_log "  Error *****:  At least one snapshot was not
deleted.\
                                           RC: $RC_DELETE_ERROR"
  +730         RC=$RC_DELETE_ERROR
  +731     fi
```

Problem Determination

In case the DB2 backup or restore command return SQL errors, the following files contain vital information. They should be analyzed in the following order:

1. The latest protocol file in the specified repository
   It contains the calls of the customer scripts with all parameters.
2. The log file of the customer script
   It is defined in the beginning of the script via variables, e.g. /tmp/gpfs_snap_BWP.log. The script also contains the variable “DEBUG”. If active, it writes useful debug information into the log. The log is appended each time the script is called. A call starts and ends with an entry like the following:

```
====================================
== Starting customer script =========
====================================
Starting doPrepare at 20130709142032 .
…
Ending doPrepare at 20130709142039 .
====================================
== Ending customer skript ===========
====================================
```

3. The DB2 diag log
   Search for strings like `sqluSnapshot` and their following entries.

Performance of GPFS Snapshots

This chapter describes the different runtimes of a GPFS-based snapshot backup compared to a traditional DB2 backup to disk. The database had no workload during the online backups.

The DB2 total backup size is approximately 60 GB.

Backup Type	**Runtime (in minutes)**
DB2 offline backup compressed	Approx. 60 mins
DB2 online backup compressed	Approx. 60 mins
DB2 offline backup with GPFS snapshot	< 2 min[1]
DB2 online backup with GPFS snapshot	< 2 min

Restore Type	**Runtime**
DB2 restore	Approx. 60 mins
DB2 restore with GPFS snapshot	Approx. 30 mins[2]

The advantage of using GPFS snapshot is obviously the faster runtime of both backup and restore compared to a traditional DB2 backup and restore. However, the GPFS snapshot backup does not provide protection from media failures because not all files are copied. The GPFS snapshot restore relies on intact file systems to restore to a previous version.

[1] during this time multiple GPFS snapshots are performed. One for each GPFS file system

[2] restore time from snapshot depends the size of the GPFS file system.

Conclusion

This part of the series demonstrated the use of IBM GPFS with the Scripted Interface and DB2 ACS to back up and restore DB2 databases using the DB2 pureScale feature. It explained the actions of all operations in detail.

Literature

[1] “ Database Installation Guide Running an SAP System on IBM DB2 10.1 with the pureScale Feature” Doc version 1.0 08/16/2012

[2] IBM “Best Practices DB2 databases and the IBM General Parallel File System™” March 2013

[3] IBM Cluster products information center → GPFS or
www.ibm.com → “Support & Downloads” → “Technical Support” → “Documentation” -> Quick Find: < enter GPFS > or
http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.gpfs.doc/gpfsbooks.html

Disclaimer and Copyrights