SQL Data Synchronization Facility

Software
Product
Research

SQL/Data Synchronization Facility Version 1.2

Table of Contents

1 Functional Description

The **SQL Data Synchronization Facility** ("**SQL/DSF"**) assists DB2/VM database administration personnel in managing **distributed** database environments. In such environments, the same data is often present at different locations in different databases. Infocenters for example, usually do not directly access the operational database tables but a copy of them. Moreover, the individual "nodes" of the distributed database may run different platforms of IBM's DB/2 RDBMS and interconnect by means of IBM's **DRDA** protocol.

In distributed database environments, it must be ensured that the distributed copies of the data remain in a consistent state. The facilities offered by SQL/DSF help database administrators to achieve this goal.

SQL/DSF offers the following functions:

Table Synchronization
Applies modifications on tables in a DB2/VM database to tables in any other database of the DB2 family.

Table Transfer
Copies entire tables from a database of the DB2 family, to tables in another DB2 database.

Table Compare
Compares tables in a database of the DB2 family, with tables in another DB2 database.

Conditional Table Transfer
Compares tables in a database of the DB2 family, with tables in another DB2 database. If the compare fails, the function initiates a table transfer from the source to the target database.

These functions are invoked:

- At the CMS prompt or from a user REXX program.
- Interactively, using the full-screen SQL/DSF user interface.
- By inserting the function request in the **SQLDSFS RULES** file of the SQL/DSF Scheduler.[1]
- By submitting an SQL/DSF **command file** to the SQL/DSF **Scheduler**.

[1]The Scheduler is a virtual machine that runs the **SQLDSFS** program, which is part of the program product.

1.1 Table Synchronization

Table Synchronization applies the data modifications performed on a table in a DB2/VM database to the same table in another database. Synchronization is a cost-effective alternative for a complete table copy, since synchronization will apply only the INSERT, DELETE and UPDATE commands executed during a DB2/VM session on the source table.

The synchronization function runs in the VM DB2/VM environment. The source table **must** be a table in a DB2/VM database. The target of the SQL/DSF SYNC function can be another DB2/VM database or any DB2 database that can be reached via DRDA, such as DB2/UDB or DB2/OS390.

Alternatively, the SYNC function can be requested to supply the SQL statements needed for synchronization on a CMS file.

1.1.1 Data Capturing

Applying data modifications from a source database to a target database implies that these changes have been captured in the source database.

The data capturing function is performed by the SQL/Auditing Facility, operating in synergy with the SQL/Monitoring Facility. The Auditor saves all table changes to the **SQL/AF audit log** file as executable SQL statements. As far as the auditor is concerned, synchronization is a particular case of auditing.

Each table to be synchronized should be defined in the Auditor's **SQLAF RULES** file with the attributes **AUDIT CHANGE** and **COLUMN *** (which is the default).[2] If the table is also subject to read auditing, AUDIT ALL should be specified. This will also audit SELECT commands, which are ignored during SYNC processing. All SQL statements, both dynamic and compiled, are intercepted regardless of their origin, that is, data capturing is also performed for accesses issued by non-DB2/VM users against DB2/VM databases.

The SQL statements captured are written to the SQL/AF audit log (a CMS file) and archived to tape by the SQL/AF Archive Processor. The SQL/AF archive tapes can be kept as long as required.

[2]Defining specific column-names as trigger for sync auditing is possible: table synchronization would then occur only when the named columns are changed.

1.1.2 Performing Synchronization

A DB2/VM table is synchronized with its target table in the target database by executing the **SQLDSF SYNC** command in the VM DB2/VM environment.

The input file for the SYNC command is the **audit log** or an audit archive tape produced by the SQL/Auditing Facility. SYNC reads all audit entries for the designated table from the audit input file, extracts the UPDATE, DELETE, INSERT, ROLLBACK and COMMIT commands and executes them against the database specified in the SQLDSF **CONNECT TARGET** command.

Rolled-back LUW's are not applied to the target. COMMIT commands are executed in the same sequence as performed in the source database. The SYNC function does not perform COMMIT on its own initiative, except for SQL errors occurring during SYNC, which result in a ROLLBACK. Although stored audit entries may have been originated by multiple concurrently active LUW's, SYNC will never process more than one LUW at a time. Therefore, only one DB2/VM agent structure is needed in the target database during synchronization.

If the target database is a non-DB2/VM database, the program relies on the DRDA capabilities of DB2/VM to execute the commands in the target database.

Note

Synchronization is performed in 2 steps:

– The first step selects from the SQL/AF audit log all log records that will be used for synchronization. The selected records are written to a CMS file, named SQLDSF WORKFILE. SQL/DSF acquires a SHARE LOCK on the table being synchronized. This ensures that no users are modifiying the table. Read-only access to the table is possible.

– The second step applies the log records from the workfile on the target database, except those records which were found rolled-back during the first step.

1.1.3 Synchronization log

All SQL statements applied to the target database are noted into a CMS file called **<target_database> DSFLOG A**. Rolled back statements also appear in the log, but their LUWid is enclosed in parentheses.

The DSFLOG file is erased at the start of synchronization. If the file should be kept, the LOGARCHIVE option should be enabled, as described on page 65.

1.1.4 Synchronization Methods

Several methods are available to indicate the tables to be synchronized:

- the **SYNC creator.tablename** command will synchronize a single table
- the **SYNC creator[%].tablename%** command will synchronize all tables satisfying the generic name in the source database
- the **SYNC TARGET creator[%].tablename%** command will synchronize all tables satisfying the generic name in the target database
- the **SYNC USING fn ft fm** command will synchronize all tables named in the specified CMS file
- the **SYNC EXCLUDING fn ft fm** option can be used on any SYNC command operating on multiple tables, to exclude named tables from the selection set

The syntax of the SYNC command is described on page 40.

1.1.5 Synchronization Mode

The SQL/AF log entries contain the expanded text of all SQL statements with all variables in the command replaced by the variable contents.[3]

For example: UPDATE EMPLOYEES SET DEPTNO = 5 WHERE EMPNO = 100

In most cases synchronization consists in executing this command against the target table in the target database. However, when the table row contains **LONG VARCHAR** columns, source table columns will be retrieved in view of synchronization. The pre-sync SELECT executed in these cases, should return **one table row** only. If the source table has a unique index, SQL/DSF will automatically use the related indexing columns in the SELECT WHERE clause. If no unique index exists, synchronization will use the source command predicates in such a way that a unique row is returned during SELECT. A logical table key can also be defined in the **SQLDSF RULES** file. For a description of the RULES syntax, see page 63.

By default, SQL/DSF will synchronize the target table in **static** SQL execution mode, if the source SQL is originated by prepped applications. To achieve this, SQL/DSF will automatically create shadow packages (named **S$nnnnnn**) in the target database during synchronization and execute the synchronizing SQL statement using the generated package. Shadow package creation also implies the creation of a small assembler program that drives the shadow package. A shadow package is maintained for each source database package that accesses the table to be synchronized.

[3]When commands access audited tables by means of **views**, the Auditor expands the view and ensures that the stored SQL statement contains real table names and real table column names only. Table or column names defined to SQL/DSF are always real names. They will automatically apply to eventual view names.

A shadow package is created :

- the first time a table is synchronized on the target
- when the source package has been reprepped.
- when the executing user does not have the shadow TEXT file on his A-disk, because the shadow package has been generated by another VM user

SQL/DSF will automatically synchronize in **dynamic** mode, without using a shadow package:

- when the original command was executed in dynamic mode (QMF, ISQL)
- when a SYNC SUBSET clause has been specified
- when an UPDATE or a DELETE with the WHERE CURRENT OF CURSOR clause or an INSERT format-2 is processed
- when a table is changed using a view
- when a command processes LONGVARCHAR or FLOAT columns
- when a command contains labelled duration expressions (such as CURRENT DATE - 1 DAY)
- when the user issues the DYNSYNC command
- when the SQLDSF OPTIONS file contains the DYNSYNC keyword

If a request is made to generate the synchronization commands on a CMS file, no SQL statements are executed on the target database. Hence, synchronization packages will not be generated.

1.1.6 Selective Synchronization

- A number of SQL/DSF statements allow to selectively synchronize a specified set of SQL/AF log entries, based on the DB2/VM LUWID, the date or the time in the log entries, and so on.

- A **SUBSET** command provides for synchronization of those SQL/AF log entries that satisfy the defined SUBSET clause in the source table. This command is useful to synchronize **global** tables into tables containing a **subset** of the global table, for example, departmental tables.

1.1.7 Synchronizing into a CMS file

If the database to be synchronized “understands” SQL but cannot be connected directly from DB2/VM using DRDA, the SQLDSF program may be requested to create a CMS file containing the text of the SQL statements necessary to synchronize the table in the target database. This file can then be submitted to the command processor of the target SQL system. Sync-to-file is assumed when an OUTFILE statement is specified for the SYNC operation. The OUTFILE statement may also specify the SQL statement terminator character expected by the target database system.

Unlike SYNC to TARGET, SYNC to OUTFILE always processes the entire audit log, without taking into account the last sync stamp for the table.

1.1.8 Scheduling synchronization

Explicit Scheduling

Synchronization is initiated by submitting the SQL/DSF SYNC command from the VM environment. Input for the command is either the SQL/AF audit log on disk or an SQL/AF archive tape. See **Synchronization Scenarios** on page 34.

Implicit Scheduling

Automated and chronological synchronization can be performed using the SQL/DSF Scheduler (see page 29).

Scheduling Controls

To allow for multiple synchronizations to the same table during a DB2/VM session, the SYNC command maintains the timestamp (the *"sync-stamp")* of the last committed synchronization LUW executed for a given table in a given target database in the SQLDSF_CONTROL_S table.

The sync-stamp is also used for restart after an SQL error on the target during synchronization. When the user has cleared the error situation on the target, restarting synchronization will automatically resume at the LUW that caused the error (and was rolled back).

The same table may be synchronized to the same target by different VM userid's. Concurrent sync of course is not allowed and will result in an error message. Moreover, when a synchronization run did terminate abruptly (e.g. by an abend), that run **must** be restarted by the same VM user that started the run.[4] Failure to do so will result in error message **SQLDSF907**.

[4]During sync, the current sync-stamp is kept on the user's A-disk in a CMS file named <database> DBSYNC. When sync comes to a controlled completion, the CMS file is transferred to the SQL table SQLDSF_CONTROL. In case of a non-controlled termination, this transfer did not occur and the last sync-stamp is on the user's A-disk.

1.1.9 Performance Considerations

Since SQL/DSF uses the **captured command** and not the DB2/VM log data for synchronization, a single UPDATE or DELETE command that alters multiple rows, will be synchronized using a single transaction.

The **static synchronization** method implemented by SQL/DSF will be considerably faster than dynamic synchronization techniques, commonly used by synchronization software. Synchronizing in dynamic mode implies that each synchronization SQL statement must be prepared by the database server before execution. Static execution simply retrieves and executes the corresponding package section, that has already been prepared. As a result, the throughput of static synchronization will be comparable to that of the originating applications.

SQL/DSF will generate the most **efficient synchronization packages** possible, even if the source package is less efficient. For example: when generating the shadow hostvar definitions, SQL/DSF will always observe the data compatibility rules, even if the source package does otherwise.[5]

1.1.10 Using SQL/DSF for table restore

Since the audit log and archives contain, in executable format, all SQL statements that altered the table, the archives can be considered as incremental backup files. Consequently, the synchronization function can be used to incrementally restore a DB2 table using an audit log or audit archive. Moreover, a number of statements (FROM_DATE, TO_DATE, FROM_TIME, TO_TIME) allow to define the scope of synchronization, thus providing support for **point in time recovery**.

1.1.11 Restrictions

SQL/DSF does not perform specific processing for tables participating in a referential constraint. It is assumed that both source and target databases have an identical constraint definition, so that SQL statements executed against the target, also update the dependent tables by the target database server. To ensure correct operations, all tables defined within the same referential constraint should be synchronized in the same run. This can be achieved by a generic tablename specification or the utilization of a USING file.

[5]When choosing a hostvar datatype that is not compatible with the column's datatype, an SQL statement may execute less efficiently, especially if the incompatibility is within the command predicate.

1.2 Table Transfer

The SQL/DSF table transfer function copies all rows of a designated table to the same table in a named target database. Like the synchronization function, table transfer is performed from the VM DB2/VM environment.

The **CONNECT SOURCE** statement identifies the database containing the table to be transferred. The **CONNECT TARGET** command designates the database that contains the target table. As for synchronization, SQL/DSF depends on the DRDA capabilities of DB2/VM to connect a non-DB2/VM database. Contrarily to synchronization, the DRDA protocol can be used for both the source and the target database. This means that SQL/DSF can transfer a DB2/VM table to OS2/DB2, but also from OS2/DB2 to DB2/VM or even from one OS2/DB2 server to another OS2/DB2 server.

SQL/DSF transfers **data only**: the target table must have been created previously in the target database. By default, a transfer deletes (with an SQL DELETE) all rows in the target at the start of the transfer. If this is not desired, the NOPURGE option of the TRANSFER command must be specified.

CMS workunits

SQL/DSF maintains two LUW's during the table transfer. One LUW is used to SELECT the rows in the source database, the other to INSERT the selected rows in the target. As a result, SQL/DSF does **not** require intermediate disk or tape storage during the transfer operation.

Selective transfer

A **subset** of the source table can be transferred to the target by specifying a **SUBSET** command. Only those source table rows satisfying the SUBSET predicate are inserted in the target.

Reformatting data during transfer

The **TRANSFER USING** option provides for a user-coded SELECT or INSERT statement to be executed during data transfer. This facility allows to transfer grouped data to the target table or to join multiple source tables into one table in the target database.

Index creation during transfer

When the transfer target is a DB2/VM database, all table indexes will automatically be dropped before transfer and recreated when transfer completes. This results in better performance than upgrading the index(es) during transfer.

1.3 Table Compare

The SQL/DSF table compare function compares all rows of a named table with the mirror table in the designated target database. Like the transfer function, table compare is performed from the VM DB2/VM environment.

The **CONNECT SOURCE** statement identifies the database containing the table to be compared. The **CONNECT TARGET** command designates the database that contains the target table. As for transfer, SQL/DSF depends on the DRDA capabilities of DB2/VM to connect a non-DB2/VM database. The DRDA protocol can be used for both the source and the target database. SQL/DSF can compare a DB2/VM table with a OS2/DB2 or MVS/DB2 table, but also the inverse.

The function compares the number of rows in both tables and the contents of each row. The compare mismatches if the number of rows is not identical, or if a difference is found in the row contents. In the latter case, the hexadecimal contents of the first mismatching rows are printed.

Contents comparing depends on the ORDER clause in the executed table SELECT, because the rows of both tables must be compared in the same sequence. SQL/DSF will attempt to determine the table order from a unique DB2/VM index or from an SQL/DSF RULES KEY statement, if present. If no such index is found or if both tables are in a DRDA database, the first table column is assumed as the unique key column.

Selective compare

Specifying the **SUBSET** command during compare, will compare a **subset** of the source and target tables.

CMS workunits

SQL/DSF maintains two LUW's during table compare. One LUW is used to SELECT the rows in the source database, the other to SELECT in the target. As a result, SQL/DSF does **not** require intermediate disk or tape storage during the compare operation.

1.4 Conditional Table Transfer

The conditional transfer function is a combination of the compare and the transfer functions. A conditional transfer compares both tables and initiates a transfer when the compare fails.

The **CONNECT SOURCE** statement identifies the database containing the table to be compared. The **CONNECT TARGET** command designates the database that contains the target table.

The function compares the number of rows in both tables and the contents of each row. The compare mismatches if the number of rows is not identical, or if a difference is found in the row contents.

Selective transfer

A **subset** of the source table can be compared and transferred to the target by specifying the **SUBSET** command. Only the source table rows satisfying the SUBSET predicate will be compared and eventually transferred.

1.5 The SQL/DSF Scheduler

SQL/DSF provides the **SQLDSFS** Scheduler program to assist an installation in setting up a **DataSync server** environment. The Scheduler includes support for:

- scheduling table synchronizations or transfers automatically and **chronologically**, as recorded in the **SQLDSFS RULES** file

- processing SQL/DSF **command files** submitted by means of the SQL/DSF SUBMIT command

Using the scheduler as an alternative for interactive SQL/DSF commands, simplifies the management of the authorities necessary for table synchronization and transfer, since these can be granted to a single userid.

The SQLDSFS function is described in detail on page 29 of this manual.

1.6 Software Prerequisites

- VM/ESA Version 1 Release 1 or later.
- DB2/VM Version 3 Release 3 or later.
- SQL/Auditing Facility Version 2 Release 1 or later. SQL/Auditing Facility is a program product available from Software Product Research.

2 Installing the Data Synchronization Facility

2.1 Pre-installation tasks

- A minidisk or SFS directory should be available to receive the SQL/DSF material. It is suggested that the SQL/AF (Auditing Facility) product disk be used for this purpose. If it is not, the SQL/AF minidisk must be accessed, when executing SQL/DSF functions.

- The Audit Processor SQLAFP must be started with the **SQLDSF** argument specified. For more information, refer to the SQL/AF User's Guide.

2.2 Installing the CMS components

This installation step loads the SQL/DSF modules, execs, help files and other material from the distribution medium to the disk or directory where the SQL/DataSync Facility has been installed (the ***SQL/DSF product disk***).

2.2.1 Prerequisites

1. Ensure that the VM userid performing the installation has write access to the SQL/DSF product disk.
2. Ensure that you have access to the SQL production minidisk in any filemode.
3. If SQL/DSF has been received on tape, ensure that a tape or cassette device has been attached to the virtual machine with virtual address 181.

2.2.2 Tape Installation

1. Access the product disk for write with CMS filemode A.
2. Issue the command: **TAPE LOAD * * A**
3. When the TAPE LOAD command has completed, you do no longer need the distribution tape.

2.2.3 Installation from other media

If SQL/DSF has been received on a medium other than tape, please follow the installation instructions present in the README file on the medium.

2.2.4 Linking the SQL/DSF Modules

1. With the product disk still accessed as "A" enter **SQLDSFPI** at the CMS prompt.
2. On the installation option screen, enter **Y** on the line **CMS Installation (Y/N)**.
3. The SQL/DSF modules are now linked on the product disk.

2.3 Installing the SQL components

SQL installation must be performed:

- In the SQL/MF log database (as defined in the **LOG DATABASE** statement of the **SQLCS CONFIG** file).
- In each database where the synchronization facility will be used. The target database may be a non-DB2/VM database, accessible via DRDA.

2.3.1 Prerequisites

1. You should have write access to the SQL/DSF product disk in filemode A.
2. You should have read access to the SQL production minidisk in any filemode.
3. Your default userid (VM username) should have DBA authority in all databases where you wish to install SQL/DSF.[6]
4. A DBspace is needed to create the synchronization tables in the SQL/MF log database. The default size of this DBspace is 1024 pages (see “SQLDSF DBspace requirements” on page 21).

2.3.2 Installing

1. Type **SQLDSFPI** at the CMS prompt.
2. On the installation option screen, enter **Y** on the line **SQL Installation (Y/N)**.
3. On the next panel, specify the name of the database for installation and the password of user SQLDBA, when initial install is being performed in the SQL/MF log database. Otherwise, the password is not required.
4. The **SQLDSF** packages are loaded in the database using a SQLDBSU RELOAD PACKAGE command under the default SQL userid (your VM userid).
5. If initial installation is being done in the SQL/MF log database, the SQL/DSF synchronization control tables will be created. At this time, the corresponding SQLDBSU command stream will be XEDITed. Specify the storage pool where the DBspace should be created. Modify the default NPAGES parameter, if the 1024 default is not suitable. Issuing XEDIT “file” will start table creation.
6. The previous panel is shown again, to allow for installation in another database. Press PF3 to terminate the installation procedure.

[6]The installation procedure does not perform a CONNECT user IDENTIFIED BY, since this is not allowed in a DRDA protocol. However, creating the SQL/DSF control tables in the SQL/MF Log database during initial product install, is done under the userid SQLDBA and the corresponding password will be requested.

2.3.3 SQLDSF DBspace requirements

The **SQLDSF** DBspace contains the tables **SQLDSF_CONTROL_P** and **SQLDSF_CONTROL_S**.

- The **SQLDSF_CONTROL_P** table has one row for each synchronization “shadow” package generated in each synchronization target database. The number of shadow packages will always be inferior to the number of packages in the synchronization source, since not all packages update synchronized tables.

- The **SQLDSF_CONTROL_S** table has one row for each table synchronized in each synchronization target database.

The average rowlength of these tables is about 60 bytes. The default DBspace of 1024 pages will allow to store about 40000 rows.

2.4 Installing the SQL/DSF Scheduler

If you intend to use the SQL/DSF scheduling facility to process SQL/DSF command files submitted by users or to execute SQL/DSF commands chronologically, you should create a virtual machine to run the **SQLDSFS** program in disconnected mode.

You should provide the scheduler with

- 16 Mb of virtual storage
- enough space on the A-disk to contain the TEXT files for the synchronization packages generated during static sync (a typical TEXT file is 4 4K blocks)
- IUCV authority to allow users to submit SQL/DSF command files
- DBA authority

In the PROFILE EXEC of the scheduler, include the ACCESS commands for the SQL/DSF, SQL/AF and SQL/MF product disks (all these products may reside on the same CMS minidisk). Initiate the scheduler using the CMS command **EXEC SQLDSFS**.

If SQL/DSF requests have to be executed chronologically, build the **SQLDSFS RULES** file on any minidisk accessed by the scheduler.

How to code the RULES file is described on page 29 of this manual.

2.5 Installing a tape mount user exit

During synchronization, SQL/DSF may be requested to get its input from an SQL/AF archive tape. Before opening a tape, SQL/DSF checks for the presence of the tape user exit **SQLAFUTX** of the SQL/AF product and calls it for tape mounting and dismounting.

How to write such an exit is explained in the SQL/AF User's Guide.

3 Invoking SQL/DSF functions

SQL/DSF functions can be invoked as follows:

- create a CMS file with the necessary SQL/DSF statements and execute the file by issuing the **SQLDSF EXEC** command
- create a CMS file with the necessary SQL/DSF statements and submit the file to the SQL/DSF Scheduler by issuing the **SQLDSF SUBMIT** command
- code the necessary SQL/DSF statements in the RULES file of the SQL/DSF Scheduler
- execute the interactive interface **SQLDSFI**.

SQL/DSF statements are divided into **declarative** statements and **executive** statements. The executive statements are **SYNC, TRANSFER, CTRANSFER, EXTRACT** and **COMPARE**. All other statements are declarative.

3.1 Execution prerequisites

The user performing SQL/DSF functions should:

- have DBA authority in the source and in the target database
- have access to the SQL/DSF, SQL/AF and SQL/MF product disks. (All these products may reside on the same CMS minidisk)
- be able to establish an R/0 link to the A-disk of the SQL/AF Audit Processor (the LINK operation itself is performed by SQL/DSF)
- be able to establish an R/0 link to the A-disk of the SQL/AF Archive Processor (the LINK operation itself is performed by SQL/DSF)
- have enough space on its A-disk for the workfile that is created during the synchronization function: (all SQL statements to be used during the current synchronization run are extracted from the SQL/AF audit log into the workfile)
- have enough space on its A-disk for the TEXT files that are part of the synchronization shadow packages (these files are not generated when performing dynamic sync)

3.2 Executing SQL/DSF command files

The SQL/DSF **EXEC** command executes SQL/DSF command files in the virtual machine of the requester.

The syntax of the EXEC command is as follows:

SQLDSF EXEC filename [filetype [filemode]]

where:

filename = filename of the CMS file containing the commands to submit
filetype = filetype of the command file [if omitted **SQLDSF** is used as default]
filemode = filemode of the command file [if omitted **A** is used as default]

Example

If the CMS file CUST1 SQLDSF A contains the statements:

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
SYNC SQLDBA.CUSTOMERS
```

to synchronize the CUSTOMERS table, the operation can be performed by using the following command on the CMS prompt or in an EXEC:

```
SQLDSF EXEC CUST1 [SQLDSF A]
```

3.3 Interactive SQL/DSF invocation

The **SQLDSFI** program allows to enter SQL/DSF requests in full-screen mode. Using the parameters entered on the input panel, SQLDSFI builds and executes an SQL/DSF command stream. Moreover, it keeps your latest invocation parameters in CMS global variables, so that your last screen input can be repeated at the next SQLDSFI invocation.

Type **SQLDSFI** on the CMS prompt.

A panel is displayed where you can enter:

- the name of a source database
- the name of a target database
- the SQL/DSF operation to perform (SYNC, TRANSFER ...)
- the name of the table(s) to be processed as **creator.tablename**

When all parameters have been entered, press ENTER. The SQL/DSF command stream will now be built and executed. After completion of the command stream, a new panel is displayed to enter a new request.

Press PF3 to terminate the SQLDSFI program.

3.4 Submitting command files to the SQL/DSF Scheduler

The SQL/DSF SUBMIT command forwards SQL/DSF command files to the virtual machine running the **SQLDSFS** program.

The syntax of the SUBMIT command is as follows:

SQLDSF SUBMIT filename [filetype [filemode]] TO server

where:

filename = filename of the CMS file containing the commands to submit
filetype = filetype of the submit file [if omitted **SQLDSF** is used as default]
filemode = filemode of the submit file [if omitted **A** is used as default]
server = name of the virtual machine running the SQL/DSF Scheduler

Submitted files usually contain SQL/DSF commands. However, CP, CMS and EXEC commands can be executed by using the prefixes CP, CMS and EXEC respectively.

The recordlength of the submit file should not exceed 256 characters. The record format can be fixed or varying-length.

Upon receipt of the file, the Scheduler executes the submitted commands immediately. If the Scheduler is busy with another submitted file or a chronological request, the new file is queued and processed in FIFO order.

If the Scheduler runs in disconnected mode, all messages issued on its console on behalf of the submitted commands are returned to the terminal of the sending user.

When an error occurs during processing of the submitted command stream, the entire stream of the current user is flushed.

Sample submit file SUB1 SQLDSF

```
CP MSG OPERATOR Transferring ACCOUNTS ...
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
TRANSFER SQLDBA.ACCOUNTS
```

The above file is submitted using the command:

SQLDSF SUBMIT SUB1 TO <servername>

4 Using the SQL/DSF Scheduler

The SQL/DSF scheduler program **SQLDSFS** runs in a disconnected virtual machine. It is started by including the command **EXEC SQLDSFS** in the PROFILE of that machine.

SQLDSFS processes submitted files and chronological scheduling requests.

4.1 Scheduling submitted files

Users may submit SQL/DSF command files to the Scheduler as described on page 28. The SUBMIT function uses the IBM IUCV protocol to forward the file to the Scheduler.

4.2 Chronological Scheduling

SQLDSFS may be requested to perform a defined command stream at a defined time of day. These specifications are recorded in a control file named **SQLDSFS RULES** which is usually, but not necessarily, located on the A-disk of the Scheduler.

Each group of commands to be scheduled must be preceded by a TIME or EVERY command.

TIME hh:mm
Specify the time of execution for the commands that follow.

EVERY n [FROM hh:mm TO hh:mm]
Specify the interval in minutes for the execution of the commands that follow. If the FROM / TO clause is specified, execution is performed during the defined period only.

When the specified time has been reached, SQLDSFS processes all commands following the TIME or EVERY command until a new TIME or EVERY command is encountered or until end-of-file. When end-of-file is found, the Scheduler pauses until midnight and then restarts itself by doing a CMS IPL. When restart is complete, the RULES are processed from begin-of-file.

Following commands can appear in the RULES:

- SQLDSF commands (such as CONNECT, SYNC, EXEC etc)
- CP commands prefixed with “CP”
- CMS commands prefixed with “CMS”
- REXX exec’s prefixed with “EXEC”
- SQL/AF commands prefixed with “SQLAFCMD”
- REXX statements (see “Embedded REXX” below)
- Comment lines prefixed with “*”

Commands appearing in the RULES file should not be enclosed in quotes.

Embedded REXX

REXX statements can be coded between the **REXX** and the **ENDREXX** statements. When the Scheduler is started, all enclosed lines are converted to temporary REXX execs, which are executed when the REXX / ENDREXX sequence is encountered.

All REXX statements are available with the facility. When SQLDSF functions are called within the embedded sequence, they should follow REXX syntax, that is, be enclosed in quotes.

Embedded REXX example

```
TIME hh:mm
REXX
    Day = DATE (‘B’) // 7          /* Get day of week */
    If Day > 4 then do             /* Weekend processing */
        ‘SQLDSF .....’
    End
    Else do
        ........
    End
ENDREXX
```

Notes

- Before processing them, the entries in the RULES file are chronologically ordered by SQLDSFS.

- When an error occurs during processing of a command, the remaining commands for the current logical function are flushed and processing continues with the next logical function, if any. For instance, if multiple SYNCs are requested at a given TIME and one of them fails, the other SYNC commands will be processed.

- When SQLDSFS is restarted manually, the first RULES entry processed will be the one with a TIME statement greater than the restart time.

- If the RULES file is modified later on and if the changes should have **immediate** effect, the scheduler machine must be restarted. To perform an orderly restart after completion of any outstanding work, issue the command:

  SSPIUCV <scheduler_VMID> SQLDSFS RESTART

  You can also rely on the **automatic restart** of the scheduler, which occurs at midnight. The RULES file will be re-examined at this time.

  The VM single console facility should be enabled for the Scheduler machine, so that errors occurring during chronological scheduling are routed to operations. Errors during processing of submitted files are automatically routed to the submitting user.

Sample SQLDSFS RULES

```
*
* Every 5 minutes in online window
*
EVERY 5 FROM 08:00 TO 18:00
CONNECT TO SOURCE DBN1
CONNECT TO TARGET DBN2
SYNC SQLDBA.Q_REQUESTS
*
* At 20h
*
TIME 20:00
CONNECT TO SOURCE DBN1
CONNECT TO TARGET DBN2
SYNC SQLDBA.CUSTOMERS
SYNC ACCOUNT.%
SQLAFCMD ARCHIVE
```

Contents of sample rules file:

- From 8 a.m. until 6 p.m. synchronize the Q_REQUESTS table every 5 minutes.
- At 20h. synchronize the customer table and all accounting tables to DBN2, using the SQL/AF disk log. After synchronization, archive the SQL/AF disk log (written to tape and cleared on disk).

5 Performing table synchronization

Prerequisites

- The table to be synchronized must have been defined previously in the SQLAF RULES file of the SQL/AF Auditing Facility with an audit option ensuring that inserts, deletes and updates are captured (for example, AUDIT ALL or AUDIT CHANGE).

- The VM userid initiating the synchronization procedure must be able to establish a read-only link to the A-disk or directory of the virtual machines that run the SQL/AF Audit and Archive Processors.

The target table must be **compatible** with the source table as follows:

- all columns defined in the source table must be defined in the target table
- all columns defined in the target, but not in the source table, should be nullable
- all column names must be identical
- all column datatypes must be identical or compatible[7]

Operation

To perform a table synchronization, following SQL/DSF commands are always required:

- a **CONNECT SOURCE** statement designating the name of the source database
- a **CONNECT TARGET** statement designating the name of the target database
- a **SYNC** or **DYNSYNC** statement naming the table to be synchronized

If SYNC must handle LONG VARCHAR columns and the source table has no unique index, the SQLDSF RULES KEY command may be required as well.

[7]A datatype is compatible when the source column can be moved to the corresponding target column. For example: a SMALLINT source column is compatible with a column defined as INTEGER in the target.

5.1 Synchronization scenarios

5.1.1 Daily synchronization at the end of a DB2 session

- Operation will be easier if the disk audit log is used. This implies that the SQL/AF log is large enough to contain all audit entries for the day, a requirement that is easier to meet when the log is in SFS. After synchronization, the disk log can be archived to tape (SQLAFCMD ARCHIVE).

- If the above requirement cannot be met, it is suggested that the disk log is archived to tape before starting synchronization with an INFILE TAPE statement. If the source database must be synchronized to several target databases, it will be faster to EXTRACT the archive tape entries for the current date to a CMS file and to specify that CMS file during sync by means on an INFILE fn ft fm.[8]

- The CCONNECT SOURCE statement should be used when the database is down.

5.1.2 Synchronization during a DB2 session

- If it is necessary to synchronize one or more tables several times a day, all input must be on the disk log, that is, explicit or implicit archiving should not occur.[9]
- Before scanning the audit log for entries to be applied, a share lock is taken on the table(s) to be synced, to prevent that audit log changes during the scan. The lock is released as soon as the scan completes.
- The share lock can be avoided as follows:
  - Issue an SQLAFCMD S_AUDIT2 command so that audit log entries are written to the secondary log.
  - Perform all synchronizations needed with the NOLOCK option.
  - Issue an SQLAFCMD S_AUDIT1 command to return to the primary log. Please note that the S_AUDIT1 command will erase the primary log before copying the secondary to the primary log.

5.1.3 Periodical (non-daily) synchronization

- Archive the current disk log to tape.
- Perform an SQLDSF SYNC with INFILE TAPE as many times as needed to cover the period since the last synchronization. This depends on the SQL/AF archive cycle. For example, if your archive cycle is weekly and you synchronize at the end of the month, at least 4 sync runs will be needed. A monthly archive cycle would require one sync run.

[8]Because the SYNC input file must be disk (for record positioning), a SYNC with an INFILE TAPE performs an implicit extract to disk. Performing such a sync several times will extract the same tape several times too.

[9]If it has, the SYNC request will abend with error message SQLDSF914, indicating that an archive tape must be processed before the disk log. Otherwise synchronization data loss would occur.

5.2 Declarative SYNC statements

Declarative statements are specified before the SYNC statement. If multiple SYNC statements occur in the executed file, the declaratives need be specified only once. However, the declarative **SUBSET** is cleared at the end of each SYNC and must be respecified, if needed.

5.2.1 CONNECT SOURCE

Syntax: **CONNECT [user IDENTIFIED BY password] TO SOURCE database**

Specify the name of the DB2/VM database that is the source for synchronization. If you do not specify ***user***, connect will be done using the default DB2 authorization-id, that is, using the VM-id. This userid should have DBA authority in the source database.

The name of the source database is used to select the auditlog entries to be applied. If synchronization involves processing of LONG VARCHAR columns, the connection is used to obtain the data values for such columns.

The CONNECT statement requires that the source database is up during synchronization processing.

5.2.2 Conditional CONNECT SOURCE

Syntax: **CCONNECT [user IDENTIFIED BY password] TO SOURCE database**

Specify the name of the DB2/VM database that is the source for synchronization. If you do not specify ***user***, connect will be done using the default DB2 authorization-id, that is, using the VM-id. This userid should have DBA authority in the source database.

The name of the source database is used to select the auditlog entries to be applied. If synchronization involves processing of LONG VARCHAR columns, the connection is used to obtain the data values for such columns.

The conditional connect statement allows to perform synchronization while the source database is down. Processing of LONG VARCHAR columns however is not possible in this case and synchronization of the corresponding table will fail. Conditional connect is not possible if the designated database is the SQL/MF log database.

5.2.3 CONNECT TARGET

Syntax: **CONNECT [user IDENTIFIED BY password] TO TARGET database**

Specify the name of the database that is the target for SYNC. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. This userid should have DBA authority in the target database, since it must be able to insert, update and delete on the target table. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

5.2.4 INFILE

Format-1: **INFILE filename filetype {filemode | A}**
Format-2: **INFILE TAPE archive-tape-label**
Format-3: **INFILE TAPE CURRENT**

The INFILE statement designates the file to be used as input for the synchronization function.

- **INFILE filename filetype filemode** designates a CMS file previously created by the "EXTRACT from archive tape" function and to be used for synchronization.

- **INFILE TAPE <label>** designates the named SQL/AF archive tape as input for the synchronization process. An implicit extract to disk will be performed before the start of synchronization. (Synchronization always requires a disk input file, because direct record read is necessary). Consequently, enough disk space should be available on the A-disk to contain all the audit records extracted from tape. SQL/DSF tape handling is explained on page 73.

- **INFILE TAPE CURRENT** performs the same functions as the format-2 statement. The current archive tape label is obtained automatically, by reading the LASTING GLOBALV file of the SQL/AF Archive Processor. An implicit extract to disk will be performed before the start of synchronization.

- If the INFILE statement is omitted, synchronization will use the disk log of the SQL/Auditing Facility **SQLAF AUDIT1**. All necessary linking and CMS access is performed by the SYNC function, provided that the executing VM userid is able to establish a read link to the A-disk or directory of the Auditor.

5.2.5 OUTFILE

Syntax: **OUTFILE filename filetype {filemode | A} [TERMINATOR x]**

The OUTFILE command should be specified only when synchronizing to a non-DRDA database. It identifies the CMS filename that will receive the text of the SQL statements needed to synchronize the target table.

The optional argument TERMINATOR specifies the command terminator character to be appended at the end of each SQL statement stored into OUTFILE.

When OUTFILE is specified, the CONNECT TARGET command should be omitted.

If the SYNC output file already exists, it will be **extended** by the current SYNC request.

5.2.6 FROM_LUW

Syntax: **FROM_LUW n**

The statement allows to start synchronization from a specified SQL LUWID. You can display the LUWID for specific log entries, using the **Log Scan** program, which is described in the SQL/AF User's Guide.

5.2.7 TO_LUW

Syntax: **TO_LUW n**

Used in combination with FROM_LUW, the statement performs synchronization for a range of SQL LUWIDs, that is, from FROM_LUW up to and including TO_LUW.

5.2.8 FROM_DATE

Syntax: **FROM_DATE expression**

The statement selects SQL/AF audit entries for a specified date. Any valid SQL date expression can be used such as: CURRENT DATE, CURRENT DATE - 1 DAY, '1995-01-10' etc. Please note that an absolute date expression should be enclosed in quotes.

5.2.9 TO_DATE

Syntax: **TO_DATE expression**

Used in combination with FROM_DATE, the statement performs synchronization for the specified date range, that is, from FROM_DATE up to and including TO_DATE. Please note that an absolute date expression should be enclosed in quotes.

5.2.10 FROM_TIME

Syntax: **FROM_TIME expression**

The statement selects SQL/AF audit entries greater than or equal to a specified time. Any valid SQL time expression can be used, such as: CURRENT TIME, CURRENT TIME - 30 MINUTES, '08.30.00' etc. Please note that an absolute time expression should be enclosed in quotes.

5.2.11 TO_TIME

Syntax: **TO_TIME expression**

Used in combination with FROM_TIME, the statement performs synchronization for the specified time range, that is, from FROM_TIME up to and including TO_TIME. Please note that an absolute time expression should be enclosed in quotes.

NOTE If more than one of the above selection statements are used, the log entries must satisfy **all** specified conditions before being selected.

5.2.12 SUBSET

Syntax: **SUBSET expression**

This command designates that SYNC is for a **subset** of the source table and identifies the subset in the expression.

A SUBSET expression has the following syntax:

<column-name> <operator> <constant>

where:
<operator> is one of the logical operators = ^= <> > < >= <=
<constant> is a string or a date expression

Coding rules

(1) Enclosing quotes are not required for alphabetic constants, unless the constant contains embedded blanks.

(2) With the exception of date expressions, no column or scalar functions are allowed in “constant”.

(3) Generically testing the leading positions of a column is requested by terminating the constant with a % sign.

(4) When testing numerical constants, the following rules should be observed:
- leading zeroes should **not** be specified
- trailing fractional zeroes **must** be specified
- negative values should have a leading **-** sign
- a decimal point is entered with a **.** sign

(5) Multiple expressions may be connected through AND / OR. These connectors should not be combined, that is, the SUBSET clause should contain only AND or OR connectors.

(6) Composite expressions must be flat: they cannot contain parentheses.

SQL/DSF evaluates the SUBSET expression as follows:

- SQL/DSF applies the SUBSET clause on INSERT commands only: synchronized DELETE and UPDATE commands are assumed to supply the SUBSET selection in their WHERE clause.
- The INSERT command is scanned for the occurrence of the SUBSET column-name in the INSERT column-list and for the occurrence in the corresponding position of the INSERT VALUES list, of a value satisfying the SUBSET condition.

The only purpose of the SUBSET command is to define a logical subset within a global table. The command should **not** be used to apply the same log entries to the same table in the same database in multiple SYNC runs. This is not possible, as a SYNC run always resumes after the last recorded sync-stamp for a given table in a given database

SUBSET command samples

```
SUBSET DEPARTMENT=10
SUBSET DEPARTMENT IN (10,11)
SUBSET INVOICE_DATE = CURRENT_DATE - 1 DAY
```

5.3 Executive SYNC statements

The **SYNC** statement extracts the DELETE, INSERT, UPDATE, COMMIT and ROLLBACK commands related to designated tables from the SQL/AF log and applies them to the target table in the target database.

Following formats are allowed for the [DYN]SYNC statement:

- **[DYN]SYNC creator.tablename(s)** : synchronizes the named source database table(s)

- **[DYN]SYNC TARGET creator.tablename(s)** : synchronizes the named target database table(s), provided that target is a DB2/VM database

- **[DYN]SYNC USING <CMS filespec>** : synchronizes the table(s) named in a CMS file

The above statement formats also provide an EXCLUDING facility and a number of command options. These common syntax elements are described separately, on page 44.

Whenever the **SYNC** command is mentioned, the **DYNSYNC** keyword can be used to force dynamic synchronization. It may be used to prevent the creation of synchronization packages on the target database.

Notes

- The SYNC command uses static SQL to perform synchronization in most cases and automatically creates SQL packages to achieve this.

- If the SYNC command completes successfully, the CMS returncode is zero. If an error occurs, the errorcode (SQLDSF**xxx**) is passed in the CMS returncode.

- When the SQL/Auditing Facility has been enabled in the target database, data changes resulting from synchronization are not audited.

5.3.1 SYNC format-1 statement

Syntax: **SYNC creator.tablename**

Performs synchronization for the named source table(s).

The specified SYNC tablename may be a **generic** name, which allows to synchronize multiple tables in a single run. The generic name is coded by using the **%** character. [10] When a generic name is used, the tablenames satisfying the generic name are obtained from the SYSCATALOG in the **source** database.

[10]SYNC will never select tables with CREATOR = SYSTEM.

5.3.2 SYNC format-2 statement

Syntax: **SYNC TARGET creator.tablename**

Performs synchronization for the named target table(s).

The specified SYNC tablename may be a **generic** name, which allows to synchronize multiple tables in a single run. The generic name is coded by using the **%** character. When a generic name is used, the tablenames satisfying the generic name are obtained from the SYSCATALOG in the **target** database.

SYNC format-2 should be used when a generic tablename would result in tablenames that do not exist in the target. For example, specifying SYNC SQLDBA.MT% requires that all MT% tables exist in the target. Specifying SYNC TARGET SQLDBA.MT% will synchronize only the MT% tables that have been created in the target.

Note

You can use a format-2 SYNC statement only when the target database is DB2/VM.

5.3.3 SYNC format-3 statement

Syntax: **SYNC USING filename filetype filemode**

Performs synchronization for the tables listed in the CMS file designated by <filename filetype filemode>. The CMS filemode may be omitted, in which case ***** is used as the default.

The CMS file should contain one line for each table to be synced. The tablename may start in any column and has the format **creator.tablename**. However, the qualifying period may be omitted and a variable number of blanks can be used to separate creator and tablename.

The specified tablename should **NOT** be generic.

5.3.4 SYNC EXCLUDING clause

The **EXCLUDING filename filetype filemode** clause designates a CMS file that contains a list of tablenames to be excluded from a table list built by a preceding clause on the SYNC statement, thus providing a “sync all but” facility.

The CMS file should contain one line for each table to be synced. The tablename may start in any column and has the format **creator.tablename**. However, the qualifying period may be omitted and a variable number of blanks can be used to separate creator and tablename. The specified tablename should **NOT** be generic.

Example

```
SYNC SQLDBA.MT% EXCLUDING MT EXCLUDE
```

where the MT EXCLUDE file contains records like:

```
SQLDBA.MT100
SQLDBA.MT130
```

5.3.5 SYNC OPTIONS clause

The OPTIONS clause on the SYNC statement has following syntax:

OPTIONS
[IGNORE_SQLCODE]
[FOR_RESTORE]
[NOLOCK]
[EXIT exitname [exit_arguments]]

- When IGNORE_SQLCODE is specified, synchronization continues after detection of an SQLCODE error. The auditlog record causing the error is written (with append) to a CMS file named **<target_database> IGNORED**. After correction of the error, the IGNORED file should be re-submitted using a SYNC command preceded by the command: **INFILE <target_database> IGNORED**. After processing, the IGNORED file must be erased manually, because of the write append operation described above.

- The **FOR_RESTORE** option should be used only when the SYNC command is used in the context of a table restore operation. FOR_RESTORE ensures that the whole audit input file is processed for restoring. When the FOR_RESTORE option is omitted, synchronization starts after the last audit log entry processed by the previous sync for the same table.

- During phase 1 of the synchronization process, a share lock is taken on the table(s) to be synchronized, to prevent that table updates are written to the SQL/AF primary audit log, while the latter is being scanned. The **NOLOCK** option disables the share lock. The option should by used only when the alternate log is active, that is, after the execution of the SQL/AF command S_AUDIT2.

- The **EXIT** option specifies the name of a statement user exit. The exit is coded as a REXX exec and is invoked for each SQL statement that will be applied to the target database during sync. The exit can modify the statement text.

  Up to 16 arguments can be passed to the exit. The arguments should adhere to the CMS tokenization rules: the length of each argument should not exceed 8 characters and should not contain blanks.

  If specified, the **EXIT** option should be the last option coded on the SYNC statement.

  How to code a statement user exit is described on page 70.

5.3.6 CHECKLOG

Syntax: **CHECKLOG**

A CONNECT SOURCE statement should precede the CHECKLOG command, to designate the source database for which the command must be performed.

A CONNECT TARGET statement should precede the CHECKLOG command, to designate the target database for which the command must be performed.

The **CHECKLOG** command compares the “last sync” timestamp for all user tables with the timestamps recorded for those tables in the SQL/AF Audit Log. If the Audit Log timestamp is higher than the “last sync” timestamp, CHECKLOG writes following items to the CMS stack:

– databasename
– table creator
– table name

If a table appears in the stack, update activity has been performed on the table in the SOURCE database and those updates have not been synchronized yet to the TARGET database.

To perform the CHECKLOG function, issue the following commands from a REXX exec (or use the embedded REXX facility of the SQL/DSF Scheduler, as described on page 30).

‘SQLDSF CONNECT TO SOURCE <databasename>’
‘SQLDSF CONNECT TO TARGET <databasename>’
‘SQLDSF CHECKLOG’

If queued() > 0 then /* some sync’s not performed */

5.3.7 CLOSE_SYNC

Syntax: **CLOSE_SYNC [creator.tablename]**

A CONNECT TARGET statement should precede the CLOSE_SYNC command, to designate the target database for which the command must be performed.

The **CLOSE_SYNC** command without tablename stores the current timestamp as the last synchronization stamp for all tables ever synchronized to the target database. If tablename is specified, the synchronization stamp is updated to the current timestamp for the named table only. As a result, synchronization data pending in the SQL/AF Audit Log, will not be processed at the next sync run.

The **CLOSE_SYNC** command should be used when all DB2 tables have been copied from the source to the target databases, by means of facilities, other than SQL/DSF.

To perform the CLOSE_SYNC function, submit the following SQLDSF commands:

'SQLDSF CONNECT TO TARGET <databasename>'
'SQLDSF CLOSE_SYNC'

5.3.8 SYNC Samples

Default SYNC procedure using the SQL/AF log:

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
SYNC SQLDBA.CUSTOMER
```

SYNC procedure using the SQL/AF archive log for the current date

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNA
INFILE TAPE SQLAF-yyyy-mm
FROM_DATE CURRENT DATE
SYNC SQLDBA.CUSTOMER
```

Synchronizing specified target tables

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
SYNC TARGET SQLDBA.MT% EXCLUDING MT EXCLUDE A
```

Generic synchronization (Sync all tables created by USERA)

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
SYNC USERA.%
```

Synchronization during a DB2 session

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
SYNC USING fn fm ft OPTIONS
```

Forcing dynamic synchronization in all cases (no sync packages created on the target)

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
DYNSYNC creator.tablename
```

Synchronization for table restore

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
INFILE TAPE SQLAF-yyyy-mm
SYNC creator.tablename OPTIONS FOR_RESTORE
```

Synchronizing into a CMS file

```
CONNECT TO SOURCE DBNA
OUTFILE SQL OUTPUT A TERMINATOR ;
SYNC SQLDBA.CUSTOMER
```

6 Using the Extract function

Prerequisites

The VM userid initiating the EXTRACT function must be able to establish a read-only link to the A-disk or directory of the virtual machine running the SQL/AF Archive Processor. This link is required when no INFILE TAPE is specified. In this case EXTRACT retrieves the tape label of the current archive from the A-disk of the Archive Processor.

Operation

The EXTRACT function copies the audit entries for a specified date or date range from the SQL/AF archive tape to a named CMS file, in view of subsequent synchronization runs. (The synchronization facility always requires a disk input file, because direct record read is necessary). EXTRACT will conduct tape attaching and mounting by sending the appropriate messages to the user.

Please note that an implicit EXTRACT occurs when an INFILE TAPE is supplied for the SYNC statement;

Invocation

The EXTRACT function is controlled by the statements FROM_DATE, TO_DATE, INFILE, OUTFILE and EXTRACT. The EXTRACT statement and the FROM_DATE are mandatory. All other statements are allowed to default.

6.1 INFILE

INFILE TAPE {archive-tape-label | CURRENT}

The INFILE statement specifies the label of the SQL/AF archive tape to be processed. This label has a standard format determined by SQL/AF, as described in the SQL/AF User's Guide.

When specifying **INFILE TAPE CURRENT**, the extract function will retrieve the tape label of the current SQL/AF archive tape from the LASTING GLOBALV file of the SQL/AF Archive Processor.

If no INFILE is specified, the function will access the A-disk of the SQL/AF Archive Processor and retrieve the label of the **current** archive tape.

6.2 OUTFILE

OUTFILE filename filetype [filemode | A}

The OUTFILE statement designates the CMS file that will receive the extracted SQL/AF log entries. This file will usually be submitted later on to a synchronization process.

If no OUTFILE is specified, the function will create a CMS file named **SQLDSF EXTRACT A**.

6.3 FROM_DATE

Syntax: **FROM_DATE expression**

The statement defines an extract date. Any valid SQL date expression can be used such as: CURRENT DATE, CURRENT DATE - 1 DAY, 1995-01-10 etc.

6.4 TO_DATE

Syntax: **TO_DATE expression**

Used in combination with FROM_DATE, the statement defines an extract date range, from FROM_DATE up to and including TO_DATE.

6.5 EXTRACT

EXTRACT [APPEND]

This statement starts the extract

- for a single date, as defined in a preceding FROM_DATE statement
- for a date range, as defined in preceding FROM_DATE and TO_DATE statements

If the **APPEND** option is present, the extracted rows are appended to an existing extract file. Otherwise an eventually existing output file is erased when extract starts.

If the command has completed successfully, the CMS returncode is zero. If an error did occur, the errorcode (SQLDSF**xxx**) is passed in the CMS returncode.

6.6 Sample

Extract the last week's log entries from the current archive tape. Using the extract, synchronize a table to 2 target databases.

```
FROM_DATE CURRENT DATE - 7 DAYS
TO_DATE CURRENT DATE
OUTFILE SQLDSF1 EXTRACT A
EXTRACT

CONNECT TO SOURCE DBN1
CONNECT TO TARGET DBN3
INFILE SQLDSF1 EXTRACT A
SYNC SQLDBA.TAB1
```

7 Transferring Tables

Prerequisites

The table to be transferred must be compatible with the receiving table in the target database, as follows:

- the number of columns must be the same in both tables
- the column order must be identical
- the column datatypes must be identical or compatible
- the target table column names are not significant, as the INSERT issued to the target during transfer does not specify a column-list

The above restrictions may not apply, when the user provides his own SELECT or INSERT statement during transfer with USING option.

After succesfull transfer, the last sync stamp for the table is set to the current timestamp. This prevents the application during the next sync, of audit log entries that were stored prior to the table transfer.

Invocation

The TRANSFER function is controlled by the statements CONNECT SOURCE, CONNECT TARGET and TRANSFER. These statements are mandatory. An optional SUBSET command may be used to transfer a specified set of table rows.

7.1 CONNECT SOURCE

CONNECT [user IDENTIFIED BY password] TO SOURCE database

Specify the name of the database containing the table to be transferred. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. Please note that in most cases, this userid should have DBA authority in the source database, since it must be able to retrieve rows from the source table. Since these accesses are done in dynamic mode, object-related privileges or the DBA privilege are required. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

7.2 CONNECT TARGET

CONNECT [user IDENTIFIED BY password] TO TARGET database

Specify the name of the database that is the target for transfer. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. Please note that in most cases, this userid should have DBA authority in the target database, since it must be able to insert into the target table. Since this is done in dynamic mode, object-related privileges or the DBA privilege are required. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

7.3 SUBSET expression

This command is used to transfer a defined set of table rows to the target. The SUBSET expression can be any SQL expression, valid in the SELECT WHERE clause for the source table. If multiple TRANSFER statements occur in the executed file, or if a multi-table TRANSFER is performed, by specifying a generic table-name, the declaratives need be specified only once. However, the declarative **SUBSET** is cleared at the end of each TRANSFER and must be respecified, if needed.

7.4 TRANSFER

TRANSFER creator.tablename [OPTIONS]
[TO creator.tablename]
[NOPURGE]
[NOSTAMP]
[NOLOCK]
[DELETE_ROWS]
[NOINDEX]
[COMMITCOUNT {100000 | n}]

In its basic form, TRANSFER copies the named table from the source database to an identically named table in the target database. If the specified table-creator and table-name is **generic**, multiple tables are transferred in a single SQL/DSF run. The generic name is coded using SQL syntax. When using generic names however, the **TO** and **NOPURGE** options cannot be specified.

Command Options

- The optional **OPTIONS** keyword is provided for compatibility with the SYNC command syntax.
- The **TO tablename** clause indicates that the tablename (or creator) of the target table is different from that in the source database.
- Before starting transfer, the target table is purged by issuing an unqualified DELETE (if SUBSET omitted) or a DELETE WHERE <SUBSET> specification. Specify the **NOPURGE** option to prevent this initial DELETE. Transfer then will **append** the target rows to the table. For large tables, it is preferable to specify NOPURGE and to clear the target table by re-creating it, prior to transfer. This will improve performance and avoid empty table pages, resulting from the DELETE.
- The DELETE that implements the pre-transfer purge may eventually fill the DB2 log (SQLCODE -933) when transferring a large table. The **DELETE_ROWS** option will prevent this, by performing a cursored DELETE with a COMMIT at regular intervals.
- The **NOINDEX** option avoids the DROP / CREATE INDEX commands that are issued by default before / after a transfer. The option is useful when tranferring a small subset to a large table.
- When the **NOSTAMP** option is specified, the last sync stamp for the table is not modified at the end of the transfer.
- The **NOLOCK** option prevents the SHARE LOCK on the source table during transfer. An EXCLUSIVE LOCK is always obtained on the target table.
- To avoid log full conditions, the transfer function performs a COMMIT after transferring <**COMMITCOUNT**> rows to the target. The default COMMITCOUNT is 100000 rows.

Notes

- The TRANSFER function does not require disk or tape storage for its operation. It uses **CMS workunits** to maintain a concurrent connection to the source and to the target database.
- If SQL/Auditing has been enabled for the table in the target database, data access resulting from TRANSFER is not audited.
- When transferring a subset of a table, the SUBSET clause is also used to DELETE the target rows before transfer. Omitting the SUBSET will result in a DELETE of the entire target table.

7.5 TRANSFER USING

TRANSFER creator.tablename USING fn ft fm
[TO creator.tablename]
[NOPURGE]
[NOLOCK]
[DELETE_ROWS]
[COMMITCOUNT {100000 | n}]

The TRANSFER USING command designates a CMS file that contains a SELECT or an INSERT format-2 statement, to be used by the transfer function.

- If the CMS file contains a SELECT, that statement will be executed by the transfer function in the source database and the resulting data will be inserted into the target table. The user-supplied SELECT must be compatible with the target table definition. The USING file may contain any valid SQL SELECT and may initiate a join, column grouping, ordering, subselecting and so on.

- If the CMS file contains an INSERT format-2 of the form INSERT INTO ... SELECT ... FROM ..., the SELECT part of the statement is executed by the transfer function in the source database and the INSERT part in the target database. This statement format allows to insert selected and named columns into the target table. All other transfer command formats issue an INSERT without column list, thereby imposing stricter compatibility rules.

When coding the USING file, following syntax rules must be observed:

- the record length of the CMS file may be fixed or varying, but should not exceed 256 bytes

- when multiple records are needed to contain the statement, a new record should be started after an SQL verb, as SQL/DSF leaves a blank between file lines, when building the executable SQL statement

- no continuation or termination mark should be provided; the statement ends at end of file

Command Options

- The **TO tablename** clause indicates that the tablename (or creator) of the target table is different from that in the source database. This clause also allows to copy a table in the same database under a new name.

- Before starting transfer, the target table is purged by issuing an unqualified DELETE (if SUBSET omitted) or a DELETE WHERE <SUBSET> specification. Specify the **NOPURGE** option to prevent this initial DELETE. Transfer then will **append** the target rows to the table. For large tables, it is preferable to specify NOPURGE and to clear the target table by re-creating it, prior to transfer. This will improve performance and avoid empty table pages, resulting from the DELETE.

- The DELETE that implements the pre-transfer purge may fill the DB2 log (SQLCODE -933) when transferring a large table. The **DELETE_ROWS** option will prevent this by performing a cursored DELETE with a COMMIT at regular intervals.

- The **NOLOCK** option prevents the SHARE LOCK on the source table during transfer. An EXCLUSIVE LOCK is always obtained on the target table.

- To avoid log full conditions, the transfer function performs a COMMIT after transferring <**COMMITCOUNT**> rows to the target. The default COMMITCOUNT is 100000 rows.

Notes

- The TRANSFER function does not require disk or tape storage for its operation. It uses **CMS workunits** to maintain a concurrent connection to the source and to the target database.

- If SQL/Auditing has been enabled for the table in the target database, data access resulting from TRANSFER is not audited.

7.6 Duplicate key handling during subset transfer

When an SQLCODE -803 occurs during transfer of a table subset, the offending row is automatically deleted on the target and the insert is retried.

To activate this facility, the SQLDSF RULES file should specify a key for the source table being transferred. This key is used by SQLDSF to perform a DELETE WHERE. The column names specified in the KEY statetement should allow for the selection of the row to be deleted.

Since detection of SQLCODE -803 depends on the existence of a unique index, the TRANSFER NOINDEX option is automatically forced during a subset transfer.

7.7 Samples

A basic transfer

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
TRANSFER SQLDBA.TAB1 COMMITCOUNT 10000
```

A multi-table transfer

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
TRANSFER ACCOUNTS.%
```

Transfer with a USING file

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
TRANSFER SQLDBA.TAB1 USING TAB1 F1 A[11]
```

Transfer with a USING file

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
TRANSFER SQLDBA.TAB1 USING TAB1 F2 A[12]
```

[11] **Contents of TAB1 F1 A:**

```
SELECT C1,SUM(C2),SUM(C3) FROM SQLDBA.TAB1 GROUP BY C1
```

[12] **Contents of TAB1 F2 A:**

```
INSERT INTO SQLDBA.TAB1(C1,C3)
SELECT C1,SUM(C3) FROM SQLDBA.TAB1 GROUP BY C1
```

7.8 Transfer performance considerations

- the transfer operation performs a COMMIT after a defined number of rows have been transferred

- before transfer, all table indexes are dropped; these indexes are recreated when transfer is complete

- the best transfer performance is obtained by the following sequence of operations:

  - drop the target table (using a drop DBspace if a DB2/VM target)
  - re-create the target table and its dependent objects
  - perform a TRANSFER NOPURGE

  Under the above conditions, transfer rates of 4000 rows per second have been measured.

  Please note that the drop and create operations can be requested from the SQL/DSF user exit SQLDSUSX, which is invoked at the start and the end of the TRANSFER statement.

8 Comparing Tables

Prerequisites

The column definitions of the tables to be compared must be compatible as follows:

- the number of columns must be the same in both tables
- the column order must be identical
- the column datatypes must be identical
- the column names are not significant

Invocation

The function is controlled by the statements CONNECT SOURCE, CONNECT TARGET and COMPARE. An SQL/DSF RULES KEY statement is required when SQL/DSF is unable to determine table sequencing automatically. An optional SUBSET command may be used to compare a specified set of table rows.

8.1 CONNECT SOURCE

CONNECT [user IDENTIFIED BY password] TO SOURCE database

Specify the name of the database containing the source table to be compared. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. Please note that in most cases, this userid should have DBA authority in the source database, since it must be able to retrieve rows from the source table. Since these accesses are done in dynamic mode, object-related privileges or the DBA privilege are required. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

8.2 CONNECT TARGET

CONNECT [user IDENTIFIED BY password] TO TARGET database

Specify the name of the database containing the target table to be compared. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. Please note that in most cases, this userid should have DBA authority in the target database, since it must be able to select from the target table. Since this is done in dynamic mode, object-related privileges or the DBA privilege are required. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

8.3 SUBSET expression

This command is used to compare a defined set of table rows. The SUBSET expression can be any SQL expression valid in the WHERE clause of the SELECT for the source and target tables. If multiple COMPARE statements occur in the executed file, or if a multi-table COMPARE is performed, by specifying a generic table-name, the declaratives need be specified only once. However, the declarative **SUBSET** is cleared at the end of each COMPARE and must be respecified, if needed.

8.4 COMPARE

COMPARE creator.tablename [TO creator.tablename]

In its basic form, the command compares the named table in the source database to an identically named table in the target database.

The **TO tablename** clause indicates that the tablename (or creator) of the target table is different from that in the source database. This allows to compare tables in the same database.

The specified table-creator and table-name may be **generic**. The generic name is coded by using the **%** character. This allows to transfer multiple tables in a single SQL/DSF run. When using generic names however, the **TO** option cannot be specified.

On exit from the compare command, the CMS returncode is zero if the tables match and **700** if they do not. In case of a generic table compare, compare stops as soon as a mismatch has been detected.

The COMPARE function does not require disk or tape storage for its operation. It uses **CMS workunits** to maintain a concurrent connection to the source and to the target database.

Mismatching column data are stored in a CMS file named **COMPARE MISMATCH A**. If such a file already exists, new mismatches are appended to the end of the file. The file must be deleted manually when the mismatch data are no longer needed.

8.5 Sample

Compare a table in two databases.

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
COMPARE SQLDBA.TAB1
```

9 Conditional Table Transfer

Prerequisites

The column definitions of the tables to be compared must be compatible as follows:

- the number of columns must be the same in both tables
- the column order must be identical
- the column datatypes must be identical
- the column names are not significant

Invocation

The function is controlled by the statements CONNECT SOURCE, CONNECT TARGET and CTRANSFER. An SQL/DSF RULES KEY statement is required when SQL/DSF is unable to determine table sequencing automatically. An optional SUBSET command may be used to compare a specified set of table rows.

9.1 CONNECT SOURCE

CONNECT [user IDENTIFIED BY password] TO SOURCE database

Specify the name of the database containing the source table to be compared. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. Please note that in most cases, this userid should have DBA authority in the source database, since it must be able to retrieve rows from the source table. Since these accesses are done in dynamic mode, object-related privileges or the DBA privilege are required. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

9.2 CONNECT TARGET

CONNECT [user IDENTIFIED BY password] TO TARGET database

Specify the name of the database containing the target table to be compared. If you do not specify ***user***, connect will be done using the default SQL authorization-id, that is, using the VM-id. Please note that in most cases, this userid should have DBA authority in the target database, since it must be able to select from the target table. Since this is done in dynamic mode, object-related privileges or the DBA privilege are required. Please note that the IDENTIFIED BY clause is valid only when connecting a DB2/VM database.

9.3 SUBSET expression

This command is used to compare - and eventually to transfer - a defined set of table rows. The SUBSET expression can be any SQL expression valid in the WHERE clause of the SELECT for the source and target tables. If multiple CTRANSFER statements occur in the executed file, the declaratives need be specified only once. However, the declarative **SUBSET** is cleared at the end of each CTRANSFER and must be respecified, if needed.

9.4 CTRANSFER

CTRANSFER creator.tablename [TO creator.tablename]

In its basic form, the command compares the named table in the source database to an identically named table in the target database. If the compare is not successful, because there is a difference in number-of-rows or row contents, a table transfer is initiated. Before starting transfer, the target table is purged by issuing an unqualified DELETE (if SUBSET omitted) or a DELETE WHERE <SUBSET> specification.

The **TO tablename** clause indicates that the tablename (or creator) of the target table is different from that in the source database.

The CTRANSFER function does not require disk or tape storage for its operation. It uses **CMS workunits** to maintain a concurrent connection to the source and to the target database.

NOTE

- If SQL/Auditing has been enabled for the table in the target database, data access resulting from TRANSFER is not audited.

- When transferring a subset table back to a global table, you **MUST** specify the SUBSET clause which is also used to DELETE the target before transfer. Omitting the SUBSET may result in a DELETE of the entire global table.

9.5 Sample

```
CONNECT TO SOURCE DBNA
CONNECT TO TARGET DBNB
CTRANSFER SQLDBA.ACCOUNTS
```

Compares the ACCOUNTS table in databases DBNA and DBNB and transfers it if mismatching.

10 SQL/DSF RULES file

The purpose of the RULES file is to supply the logical key for tables without a unique index. This specification may be needed during SYNC[13], COMPARE or CTRANSFER commands. The specification is also used to provide for automatic delete of duplicate keys during a subset transfer.

The **SQLDSF RULES** file should be placed on a minidisk that is accessible to all virtual machines performing SQL/DSF functions. You may choose to place the file on the SQL/DSF product minidisk.

A logical definition consists of a **DATABASE**, a **TABLE,** and a **KEY** command. The syntax of these commands is as follows:

DATABASE nnnnnnnn

Specify the name of the source database containing the table.

TABLE creator.table-name

Specify the creator and tablename for which the logical key is being defined.

KEY column-name [,column-name-2, ... column-name-n]

Specify one or more column-names that make up the logical table key. Multiple column-names must be separated by a comma. The column-names specified are not necessarily indexing columns.

[13]Only when commands processing LONG VARCHAR columns are synchronized.

11 SQL/DSF OPTIONS file

The SQLDSF OPTIONS file may specify following execution options:

ARCHPROC
Specifies the name of the virtual machine running the SQL/AF Archive Processor, if the default VMID of SQLAFARC is not used.

DYNSYNC
Forces dynamic synchronization in all cases, even when users request static sync.

LOGARCHIVE ON|OFF
LOGARCHIVE ON (which is the default value) requests that the synchronization logfile **<databasename> DSFLOG** be added to the archive file **<databasename> DSFLOGA** at the end of synchronization. It is the users responsability to erase the DSFLOGA file when no longer needed.

Each option should be coded on a separate line. The keyword may begin in any column. If the option has multiple keywords, a variable number of blanks are used to separate the keywords.

The SQLDSF OPTIONS file is accessed using the CMS search chain. If the file is stored on the SQL/DSF product disk, it will apply for all SQL/DSF users (unless they have a private SQLDSF OPTIONS file).

Example of an SQLDSF OPTIONS file

```
DYNSYNC
ARCHPROC XYZ
```

12 PROGRESS command

During an SQL/DSF synchronization run, the **PROGRESS** immediate command shows the number of audit log records to be processed in the current run and the number and percentage of log records already processed.

The PROGRESS command is entered on the console of the virtual machine that runs the SYNC function.

13 SQL/DSF User Exits

13.1 Process User Exit

At the begin and the end of a table synchronization or table transfer operation, the user exit SQLDSUSX is invoked with the following parameters:

- invocation_call_type a character string with one of the following values
  - begin_sync
  - end_sync
  - begin_transfer
  - end_transfer
- source_database for sync or transfer
- target_database for sync or transfer
- creator of source table for sync or transfer
- name of source table for sync or transfer

The user exit may perform all necessary processing, including DB2 access.

When performing transfer for a generic tablename, the exit is invoked twice for each table included by the generic name.

A dummy SQLDSUSX EXEC is delivered with SQL/DSF.

13.2 Record User Exit

If present, the SQLDSFUX user exit will be invoked for each audit logrecord processed during table synchronization. The SQLDSFUX EXEC is written using the same conventions as the SQLAFUXP EXEC.

SQLDSFUX should start with a call to the SQLAFUXP module, which will return all fields of the current audit record as REXX variables, with the same names as for SQLAFUXP. Please refer to the SQL/AF User's Guide for a description of SQLAFUXP.

SQLDSFUX should exit with a returncode 0 if SQL/DSF should apply the current logrecord. A returncode different from 0 will bypass SQL/DSF apply processing for the record.

SQLDSFUX can be used to provide local synchronization apply logic.

13.3 Statement User Exit

The statement user exit is invoked during table synchronization. The exit can inspect and modify each SQL statement executed during synchronization.

There is no predefined name for the statement user exit and an installation may have multiple exits. The SYNC statement names the exit to be invoked for a given synchronization run, as described on page 45.

When the exit is invoked, it receives following arguments:

- name of the source database
- name of the target database
- pointer to the SQL statement that will be applied
- length of the SQL statement
- if specified, the arguments from the SYNC EXIT statement

The exit uses the REXX **STORAGE** builtin function to retrieve the SQL statement and to modify it.

To facilitate word scanning of the SQL statement, it is passed in normalized format, that is, with a blank between each syntactical element.

For example:

the statement UPDATE TEST SET COL1='XXX' WHERE COL2='ZZZ'
is passed as UPDATE TEST SET COL1 = 'XXX' WHERE COL2 = 'ZZZ'

The exit should pass a returncode as follows:

- If the statement should not be synchronized to the target database, a negative value should be returned.
- If the statement has not been modified by the exit, a zero value should be returned.
- If the statement has been modified, the current length of the statement should be passed on exit.

Before calling the exit, SQL/DSF establishes a separate CMS workunit, which can be used by the exit to execute SQL statements. No database connection exists when the exit is entered.

Sample Statement Exit

```
/*
     Sample synchronization exit


 Entry arguments :  source database
                    target database
                    pointer to SQL statement that will be synced
                    length of statement

 Returncode      :  < 0 if statement should not be synced
                    = 0 if statement must be synced without change
                    > 0 if a changed statement must be synced
                        with the length specified in returncode
*/

 address command

 arg source_db target_db statement_pointer statement_length

 return_value = 0         /* assume statement applied without change */

/* Get the SQL statement into the variable "statement" */

 statement=storage(statement_pointer,statement_length)

 if <statement should not be applied on target> then return_value=-1

 else

/*
    If needed, modify the "statement" variable.

    Modifying the statement text for SQL/DSF is done as follows:

    x=storage(statement_pointer,length(statement),statement)
    return_value=length(statement)

*/

 exit return_value
```

14 Tape Handling

14.1 User tape handling exit

During processing of an SQL/AF archive tape, as requested by an INFILE TAPE statement, SQL/DSF will call the user exit **SQLAFUTX EXEC**, if it is found on the CMS search chain. The purpose of the exit is to integrate the installation's tape management software.

- An **EXEC SQLAFUTX OPEN <tapelabel>** command is issued before SQL/DSF opens the archive tape.

- An **EXEC SQLAFUTX CLOSE <tapelabel>** is issued when SQL/DSF closes the archive tape.

14.2 Default tape handling

If no user tape handling has been provided during tape open, SQL/DSF will issue a message to request the attaching of a tape device and the mounting of the tape. If SQL/DSF runs in a disconnected machine, these messages should be routed to operations via SCIF.

The FILEDEF and LABELDEF commands required during tape open, are issued by SQL/DSF.

15 SQL/DSF Control Tables

Following DB2/VM tables are maintained by SQL/DSF to control its operations:

15.1 SQLDSF_CONTROL_P

Column	Datatype	Contents
DBN	CHAR(8)	name of sync target database
PCREATOR	CHAR(8)	creator of the package accessing the synchronized table
PNAME	CHAR(8)	name of the package accessing the synchronized table
SNAME	CHAR(8)	name of the generated synchronization shadow package (S$000001 to S$999999)
PDATE	TIMESTAMP	timestamp when shadow package has been generated

Controls the generation of synchronization shadow packages. For all synchronization target databases, a row is inserted for every table and source package performing insert, update or delete on the synchronized table.

15.2 SQLDSF_CONTROL_S

Column	Datatype	Contents
TCREATOR	CHAR(8)	creator of the synchronized table
TNAME	CHAR(18)	name of the synchronized table
DBN	CHAR(8)	name of sync target database
OWNER	CHAR(8)	VM userid performing SYNC - blank if no sync in progress
SYNCTOD	CHAR(8)	timestamp of last audit log record synchronized to target database (in hardware clock format)

Maintains synchronization control information for each source table processed in each target database. A row is inserted for every table synchronized on each target database.

15.3 SQLDSF_CONTROL_O

Column	Datatype	Contents
DBN	CHAR(18)	name of sync source database
OPEN_LUWID	INTEGER	open LUWid
OPEN_RECNO	INTEGER	first recno on audit log
OPEN_AGENT	INTEGER	agent number
OPEN_STAMP	CHAR(8)	LUW timestamp

Maintains information for each LUW that was not committed when initiating synchronization.

16 Messages

SQLDSF001 Syntax error in the above statement.

When a syntax error is detected, the statement in error is displayed and another error message may follow.

Action Correct the statement in error and resubmit the SQL/DSF command stream. When entering SQL/DSF commands from the CMS prompt, it may be necessary to issue the SQLDSF END command to clear pending command arguments.

SQLDSF101 SQLCODE validating date/time expression is *n*

An invalid date or time expression was entered on the commands FROM_DATE, TO_DATE, FROM_TIME or TO_TIME.

n is the SQLCODE received during expression evaluation

Action Correct the statement in error and resubmit the SQL/DSF command stream.

SQLDSF102 SQLCODE connecting target database is *n*

n is the SQLCODE received when attempting to connect the target database during SYNC, COMPARE or TRANSFER.

Action Examine the SQLCODE. Ensure that the target database is active and that you have the privileges to connect it.

SQLDSF103 An SQL error has occurred

A negative SQLCODE has been received after an SQL statement during SYNC, COMPARE or TRANSFER. Messages SQLDSF104 to SQLDSF108 will further explain the error.

SQLDSF104 SQLCODE *n*
SQLERRM *nnn*
SQLERRD1 *n*

The above SQLCA fields explain the SQL error signalled by message SQLDSF103.

Action Use the above data to determine the nature of the SQL error.

SQLDSF105 SQLSTATE *n*

The above item explains the SQL error signalled by message SQLDSF103.

Contrarily to the SQLCODE, which slightly varies across IBM/DB2 platforms, the SQLSTATE value is unambiguous in all these systems.

SQLDSF106 LUWID *n*

When the SQL error occurs during a SYNC operation, the number of the originating LUW in the source database is shown.

SQLDSF107 SYNCSTAMP *n*

When the SQL error occurs during a SYNC operation, the message shows the SQL/AF stamp for the failing command. The syncstamp is identical for all commands in a given SQL LUW. It represents the hexadecimal value of the hardware clock at the begin of the LUW.

SQLDSF108 SQL Command Text:

When the SQL error occurs during a SYNC operation, the message shows the original command text causing the error.

SQLDSF110 SQLCODE connecting source database is *n*

n is the SQLCODE received when attempting to connect the source database during SYNC, COMPARE or TRANSFER.

Action Examine the SQLCODE. Ensure that the source database is active and that you have the privileges to connect it.

SQLDSF111 KEY command is required

During COMPARE or CTRANSFER, the KEY specification has been omitted and SQL/DSF was unable to retrieve a key definition from a unique index in SYSINDEXES. During SYNC, a unique index was not available and the pre-SYNC SELECT resulted in the retrieval of more than one table row.

Action A KEY command must be supplied in the SQLDSF RULES file.

SQLDSF112 SQLCODE connecting SQL/MF database is *n*

n is the SQLCODE received when attempting to connect the SQL/MF database during static SYNC.

Action Examine the SQLCODE. Ensure that the SQL/MF database is active and that you have the privileges to connect it.

SQLDSF113 SQLCODE selecting SQL/MF STMNTS table is *n*

n is the SQLCODE received when selecting the program command text from the SQL/MF SQLCS_SQL_STMNTS table during static SYNC.

Action Examine the SQLCODE. Ensure that the SQL/MF database is active and that you have the necessary SQL privileges..

SQLDSF114 SQLCODE selecting SQLDSF_CONTROL is *n*

n is the SQLCODE received when accessing the SQLDSF_CONTROL table during static SYNC.

Action Examine the SQLCODE. Ensure that the SQL/MF database is active and that you have the necessary SQL privileges to access the control table.

SQLDSF115 SQLCODE inserting SQLDSF_CONTROL is *n*

n is the SQLCODE received when accessing the SQLDSF_CONTROL table during static SYNC.

Action Examine the SQLCODE. Ensure that the SQL/MF database is active and that you have the necessary SQL privileges to access the control table.

SQLDSF116 Syntax error in SQLDSF RULES dataset

A syntax error was detected when reading the RULES dataset to obtain the table key during SYNC or COMPARE.

Action Correct the RULES statement in error.

SQLDSF117 CMS error n writing DBSYNC file

A CMS error has occurred when reading the DBSYNC input file. For an explanation of the returncode, please refer to **CMS File System Returncodes** on page 91.

SQLDSF118 Target table should be different from source

If the source and target database are identical, a transfer operation should specify different tablenames, as a table cannot be copied to itself.

SQLDSF119 SQLCODE -nnn inserting SQLDSF_CONTROL_P

An SQL error has occurred when maintaining the shadow package control table. Examine the SQLCODE for problem determination.

SQLDSF201 CMS file system error. Returncode is *n*

A CMS error has occurred when reading an SQL/DSF input file. For an explanation of the returncode, please refer to **CMS File System Returncodes** on page 91.

SQLDSF400 EXTRACT from <tape-label> in progress

Informatory message issued at the start of an EXTRACT operation.

SQLDSF401 Open error on SQL/AF tape archive

An error has occurred when opening the archive tape during EXTRACT.

Action Examine the preceding CMS messages to determine the nature of the error.

SQLDSF402 CMS file system error on OUTFILE. Returncode is *n*

A CMS error has occurred when writing the CMS output file during an EXTRACT or SYNC-to-file operation. For an explanation of the returncode, please refer to **CMS File System Returncodes** on page 91. The most common returncode for output is 13, meaning that there is not enough space on the minidisk or directory written to.

SQLDSF403 Archive tape label could not be located

During an EXTRACT or SYNC operation, an INFILE TAPE CURRENT command was supplied and the attempt of SQL/DSF to retrieve the label of the current SQL/AF archive tape has failed.

Action Ensure that the executing virtual machine is able to establish a read link to the A-disk or directory of the SQL/AF Archive Processor.

SQLDSF404 Please attach a tape drive to <*userid*> as 181

Message issued during the tape mount logic when extracting audit log records from an SQL/AF archive tape.

Action Attach a tape drive to your virtual machine. Processing continues when this has been done.

SQLDSF405 Mount <*archive-tape-label*> archive tape on 181

Message issued during the tape mount logic when extracting audit log records from an SQL/AF archive tape.

Action Mount the tape labelled ***<archive-tape-label>*** on the attached drive. Processing continues when the tape has been mounted.

SQLDSF409 EXTRACT completed

Informatory message issued at the completion of an EXTRACT operation.

SQLDSF601 SUBMIT file does not exist

The file specified on the SQLDSF SUBMIT command does not exist or cannot be accessed.

SQLDSF602 CMSIUCV rc *n* connecting to *nnn*

The IUCV CONNECT command that initiates file submission to the SQL/DSF server has failed.

Action Ensure that you have the necessary IUCV privileges to connect to the server and that the server is active. For an explanation of the IUCV returncode, please refer to **IUCV Returncodes** on page 89.

SQLDSF603 Timeout during IUCV CONNECT to *nnn*

The IUCV CONNECT command that initiates file submission to the SQL/DSF server did not complete within the timeout interval.

Action Ensure that the SQL/DSF server is active.

SQLDSF604 IUCV rc *n* on SEND path to *nnn*

The IUCV SEND command that submits the file to the SQL/DSF server has failed.

Action Ensure that the server is active. For an explanation of the IUCV returncode, please refer to **IUCV Returncodes** on page 89.

SQLDSF605 Timeout during IUCV SEND to *nnn*

The IUCV SEND command that performs file submission to the SQL/DSF server did not complete within the timeout interval.

Action Ensure that the SQL/DSF server is active.

SQLDSF606 HNDIUCV rc is *nnn*

The SQLDSF SUBMIT function was unable to establish an IUCV interrupt handler.

Action Note the returncode and call for software support.

SQLDSF701 Compare is in progress

Informatory message issued at the start of a COMPARE command.

SQLDSF702 *n* FROM rows compared

Informatory message showing the progress of the compare operation. The message is issued each time 100000 table rows have been compared.

SQLDSF703 *n* FROM rows compared

Informatory message issued at the end of compare. It shows the total number of rows compared on the source table.

SQLDSF704 Tables do compare

Informatory message indicating that both tables are identical.

SQLDSF705 Tables do not compare

Informatory message indicating that the compared tables are not identical, either because their rowcount does not match or because there is a difference in row contents. In the latter case, the hexadecimal contents of the first mismatching row are printed. This is not done during a CTRANSFER command, in which case the message triggers a table transfer.

Action Ensure that both tables are processed in the same sequence. If needed, supply a KEY command.

SQLDSF706 *n* rows in table *nnn*

If the tables do not match, the message is issued twice, showing the number of rows in both the source and the target table.

SQLDSF707 Number of mismatching rows is *nnn*

If the tables do not match, the message shows how many table rows have a different content.

SQLDSF801 Transfer is in progress

Informatory message issued at the start of a TRANSFER command.

SQLDSF802 *n* rows transferred

Informatory message showing the progress of the transfer operation. The message is issued each time 100000 table rows have been transferred.

SQLDSF896 *n* duplicate rows deleted on target

A duplicate key condition (SQLCODE -803) occurred *n* times during table transfer. Using the KEY specification in SQLDSF RULES, these duplicate rows were removed.

SQLDSF897 TRANSFER to target *nnn* successfully completed

Informatory message issued at the normal completion of a TRANSFER operation to database ***nnn***.

SQLDSF898 TRANSFER to target *nnn* abnormally completed

Informatory message issued at the abnormal completion of a TRANSFER operation to database ***nnn***. Another message will precede and explain the nature of the error.

SQLDSF899 *n* rows transferred

Informatory message issued at the completion of a TRANSFER operation. It shows the total number of rows transferred.

SQLDSF901 SYNC is in progress

Informatory message issued at the start of a SYNC command.

SQLDSF902 ***n*** **SYNC requests processed**

Informatory message showing the progress of the synchronization operation. The message is issued after processing 10000 SQL/AF log entries.

SQLDSF904 **CMS RC *n* writing shadow package input file**

During static SYNC a shadow program needed to be created. When writing the source text of the package to the A-disk before prep, the CMS file system returned error **n**.

For a list of CMS errorcodes, see page 93. Errorcode 13 indicates that there is not enough space on the A-disk to write the source text.

SQLDSF905 **RC *n* prepping shadow package input file *nnnnnn***

During static SYNC a shadow program needed to be created. When prepping the source text of the package, the prep and compile exec SQLDSFPR returned an error. The name of the shadow package shown will have the format **S$nnnnnn**.

Action You should attempt to determine the nature of the error by examining the files S$nnnnnn LISTPREP (the prep listing) and/or S$nnnnnn LISTING (the assembly listing).

If you cannot recover from the error, you should save the above files together with the source input file S$nnnnnn ASMSQL and call for software support.

SQLDSF906 **Pre_sync SELECT returns more than one row**

A synchronization was attempted for a command referencing LONG VARCHAR columns. In such cases a pre-sync SELECT on the source table is required. This SELECT used the specified KEY command clause which did not allow retrieval of a unique row.

Action Modify the KEY clause so that it designates a unique table key. If this is not possible, you cannot SYNC this table (but you can TRANSFER it).

SQLDSF907 **Synchronization process currently owned by VM userid *nnn***

A synchronization run for the table has already been started by the named VM user. Concurrent synchronization is not allowed.

If a previous synchronization run came to a non-controlled completion, the run must be restarted by the VM userid **nnn.**

SQLDSF908 **USING [EXCLUDING] File** fn ft fm **does not exist.**

The CMS file named in the USING or the EXCLUDING clause does not exist.

SQLDSF909 **Scanning for rolled-back and open LUW's**

Informatory message issued at the begin of the synchronization run. Messages SQLDSF910 and SQLDSF911 show the result of the scan.

SQLDSF910 Rolled-back LUW's found during scan: n

The sync scan found <n> rolled-back LUW's on the audit input file. These LUW's are not processed during sync.

SQLDSF911 Open LUW's found during scan: n

The sync scan found <n> LUW's that are in progress. They will not be processed during the current sync run. Information concerning these LUW's is saved so that these LUW's can be processed during the next synchronization run.

SQLDSF912 Processing previously open LUW's

Informatory message issued when previously open LUW's are being processed.

SQLDSF913 Waiting on audit log quiesce ...

The SQL/AF audit processor is processing queued audit requests. Synchronization will continue when these queued requests have been stored to the audit log.

SQLDSF914 Audit archive records pending for sync

The audit log was archived to tape and a request is made to synchronize from the current (disk) audit log. Synchronization must be performed using the audit archive before processing the disk log. If you are sure that all audit entries have been processed, delete the file **SQLAF ARCHIVED** on the A-disk of the SQL/AF Audit Processor machine.

SQLDSF915 User invoking SQL/DSF SYNC must have DBA authority

Users invoking synchronization must be DBA. This is checked by SQL/DSF in the source database and in the target database, if the latter is a DB2/VM server.

SQLDSF993 ***n*** **sync transactions executed in dynamic mode**

Informatory message issued at the completion of a SYNC operation. It shows the number of synchronization transactions executed in dynamic mode, without using a synchronization shadow package. Usually, these transactions were also executed dynamically in the source database (e.g. QMF). SQL/DSF may automatically opt for dynamic execution, for example, when long varchar or floating-point columns are being processed.

SQLDSF994 ***n*** **target rows deleted**

Informatory message issued at the completion of a SYNC operation. It shows the number of rows deleted from the target table due to synchronization.

SQLDSF995 ***n*** **target rows inserted**

Informatory message issued at the completion of a SYNC operation. It shows the number of rows inserted into the target table due to synchronization.

SQLDSF996 ***n*** **target rows updated**

Informatory message issued at the completion of a SYNC operation. It shows the number of rows updated on the target table due to synchronization.

SQLDSF997 **SYNC to target *nnn* successfully completed**

Informatory message issued at the normal completion of a SYNC operation to database ***nnn***.

SQLDSF998 **SYNC to target *nnn* abnormally completed**

Informatory message issued at the abnormal completion of a SYNC operation to database ***nnn***. Another message will precede and explain the nature of the abnormal termination.

SQLDSF999 **Number of log records scanned: n**

Informatory message indicating how many audit log records were read during the SYNC process.

17 IUCV Returncodes

For a complete description of the IUCV returncodes, refer to the IBM manual **CP Programming Services** (SC24-5520-02) in the chapter **IUCV Function Descriptions**. Use the last 2 digits of the IUCV code displayed to locate the IPRCODE in the manual.

The recoverable IUCV error codes are the following:

1011 The target communicator is not logged on.
1012 The target communicator is logged on but is not enabled for IUCV.
1013 Maximum number of IUCV connections for the source communicator exceeded. Specify a larger MAXCONN in the VM/ESA directory.
1014 Maximum number of IUCV connections for the target communicator exceeded. Specify a larger MAXCONN in the VM/ESA directory.
1015 The source communicator is not authorized (in the VM directory) to connect to the target communicator

For all other returncodes, call for software support.

18 CMS File System Returncodes for Read

1 File not found, disk not accessed, or insufficient authority.

2 Invalid buffer address.

3 I/O operation to a minidisk failed.

4 First character of file mode is illegal.

5 Number of records to read is equal to zero.

7 AFT is not marked with a record format of F or of V. If the file was not previously opened, this indicates that the file has an invalid record format.

8 Successful operation, but the buffer was too small to hold all of the requested data.

11 Number of records to read is not exactly one for a file with variable-length records.

12 No records were read because end of file was reached or because the position parameter specified a record number greater than the number of records in the file.

13 Found an invalid displacement in the AFT for a file with variable-length records (this indicates a coding error: it should not occur).

20 Invalid character detected in file name.

21 Invalid character detected in file type.

25 Insufficient free virtual storage available for file system control blocks

26 Position is negative, the number of records to read is negative, or position plus the number of records to process exceeds the file system capacity.

29 Storage group space limit reached.

30 Some error, other than those in this list of codes, occurred while accessing an SFS file. No rollback occurred.

31 Rollback occurred while trying to access an SFS file. The work unit ID on which the rollback occurred is the default work unit ID at the time the file was opened by the first operation to the file.

40 One of the following errors occurred:

A required CSL routine was dropped.
A required CSL routine was not loaded.
There was an error in a user exit routine.
There was an error calling the user accounting exit routine

42 The variable length record read is invalid.

48 File is empty.

49 External object cannot be opened.

50 File is in DFSMS/VM migrated status and implicit RECALL is set to OFF.

51 Error occurred during DFSMS/VM file recall processing.

55 APPC/VM error.

70 SFS file sharing conflict or minidisk file is already open by DMSOPEN or DMSOPDBK with an output intent.

80 I/O error accessing OS dataset.

81 OS read password protected dataset.

82 OS dataset organization is not BSAM, QSAM, or BPAM.

83 OS dataset has more than 16 extents.

84 Attempt to read a file on an OS or DOS formatted minidisk.

99 A required system resource is unavailable for one of the following reasons:

There is insufficient virtual storage for the file pool server.
The file pool server is unavailable.
File is in migrated status and DFSMS is not enabled.

IBM Source: CMS Application Development for assembler
SC24-5453-02

19 CMS File System Returncodes for Write

1 Not authorized to write to file.

2 Invalid buffer address.

3 I/O operation to a minidisk failed.

4 First character of file mode is illegal or disk not accessed.

5 Second character of file mode is illegal.

6 The last record number to be written is too large (more than 65535) to fit in a halfword and an extended plist is not specified.

7 Position specifies a record number that is more than one greater than the current number of records in a file with variable-length records.

8 Size of output buffer is not greater than zero or an attempt was made to write a null record to a file with variable length records.

11 FSCB is not marked with a record format of F nor of V.

12 Disk or directory not accessed R/W.

13 Disk is full.

14 Size of output buffer is not evenly divisible by the number of records for a file with fixed-length records.

15 Attempt to alter the record length of a file with fixed-length records.

16 Record format specified not the same as file.

17 Size of output buffer is greater than 65535 for a file with variable-length records.

18 Number of records to write is not exactly one for a file with variable-length records.

20 Invalid character detected in file name.

21 Invalid character detected in file type.

25 Insufficient free storage available for file system control blocks

26 Position specifies a negative record number or number of records to write is negative or position plus the number of records exceeds the file system capacity ($2^{31} - 1$) or logical block number computed by system exceeds the file system capacity ($2^{31} - 1$).

29 The storage group space limit was reached.

30 Some error, other than those in this list of codes, occurred while accessing an SFS file. No rollback occurred.

31 Rollback occurred while trying to access an SFS file.

38 File explicitly opened with read intent.

39 A disk is accessed as a read only extension of another, and a given file exists on the extension disk but not on the parent disk.

40 One of the following errors occurred:

A required CSL routine was dropped.
A required CSL routine was not loaded.
There was an error in a user exit routine.
There was an error calling the user accounting exit routine

49 External object cannot be opened.

50 File is in DFSMS/VM migrated status and implicit RECALL is set to OFF.

51 Error occurred during DFSMS/VM file recall processing.

55 APPC/VM error.

70 One of the following sharing conflicts occurred:

The file is locked.
The file pool server detected a deadlock.
The file is open for write through SFS OPEN.
The file is open for write by another user.
You attempted to write to a file that is currently implicitly open for READ, but the file has been changed since it was originally opened.
The minidisk file is already open by DMSOPEN or DMSOPDBK when issuing an FSWRITE.

80 I/O error accessing OS dataset.

81 OS read password protected dataset.

82 OS dataset organization is not BSAM, QSAM, or BPAM.

83 OS dataset has more than 16 extents.

84 Attempt to write a file on an OS or DOS formatted minidisk.

99 A required system resource is unavailable for one of the following reasons:

There is insufficient virtual storage for the file pool server.
The file pool server is unavailable.
File is in migrated status and DFSMS is not enabled.

IBM Source: CMS Application Development for assembler
SC24-5453-02