IBM® DB2 Universal Database™

Administrative API Reference

Version 8

SC09-4824-00

IBM® DB2 Universal Database™

Administrative API Reference

Version 8

SC09-4824-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

Contents

About This Book

This book provides information about the use of application programming interfaces (APIs) to execute database administrative functions. It presents detailed information on the use of database manager API calls in applications written in the following programming languages:

- C
- C++
- COBOL
- FORTRAN
- REXX.

For a compiled language, an appropriate precompiler must be available to process the statements. Precompilers are provided for all supported languages.

Who Should Use this Book

It is assumed that the reader has an understanding of database administration and application programming, plus a knowledge of:

- Structured Query Language (SQL)
- The C, C++, COBOL, FORTRAN, or REXX programming language
- Application program design.

Chapter 1. Application Programming Interfaces

This section describes the DB2 application programming interfaces in alphabetical order. The APIs enable most of the administrative functions from within an application program.

Note: Slashes (/) in directory paths are specific to UNIX based systems, and are equivalent to back slashes (\) in directory paths on Windows operating systems.

DB2 APIs

The following tables show the DB2 APIs with the DB2 samples. The first table lists the DB2 APIs grouped by functional category, their respective include files, and the sample programs that demonstrate them (See the note after the table for more information on the include files). The second table lists the C/C++ sample programs and shows the DB2 APIs demonstrated in each C/C++ program. The third table shows the COBOL sample programs and the DB2 APIs demonstrated in each COBOL program.

DB2 APIs, Include files, and Sample Programs
Table 1.

C/C++ Sample Programs with DB2 APIs
Table 2 on page 9.

COBOL Sample Programs with DB2 APIs
Table 3 on page 12.

Table 1. DB2 APIs, Include files, and Sample Programs

DB2 API	Include File	Sample Programs
Database Manager Control		
db2DatabaseQuiesce - Database Quiesce	db2ApiDf	n/a
db2DatabaseUnquiesce - Database Unquiesce	db2ApiDf	n/a
db2InstanceStart - Instance Start	db2ApiDf	C: `instart.c` C++: `instart.C`
db2InstanceStop - Instance Stop	db2ApiDf	C: `instart.c` C++: `instart.C`
db2InstanceQuiesce - Instance Quiesce	db2ApiDf	n/a
db2InstanceUnquiesce - Instance Unquiesce	db2ApiDf	n/a

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
sqlesdeg - Set Runtime Degree	sqlenv	C: `ininfo.c` C++: `ininfo.C`
Database Control		
db2DatabaseRestart - Restart Database	db2ApiDf	C: `dbconn.sqc` C++: `dbconn.sqC`
sqlecrea - Create Database	sqlenv	C: `dbcreate.c` `dbrecov.sqc` `dbsample.sqc` C++: `dbcreate.C` `dbrecov.sqC` COBOL: `db_udcs.cbl` `dbconf.cbl` `ebcdicdb.cbl`
sqlecran - Create Database at Node	sqlenv	n/a
sqledrpd - Drop Database	sqlenv	C: `dbcreate.c` C++: `dbcreate.C` COBOL: `dbconf.cbl`
sqledpan - Drop Database at Node	sqlenv	n/a
sqlemgdb - Migrate Database	sqlenv	C: `dbmigrat.c` C++: `dbmigrat.C` COBOL: `migrate.cbl`
db2XaListIndTrans - List Indoubt Transactions	db2ApiDf	n/a
sqle_activate_db - Activate Database	sqlenv	n/a
sqle_deactivate_db - Deactivate Database	sqlenv	n/a
sqlcspqy - List DRDA Indoubt Transactions	sqlxa	n/a
Database Manager and Database Configuration		
db2CfgGet - Get Configuration Parameters	db2ApiDf	C: `dbinfo.c` `dbrecov.sqc` `inauth.sqc` `ininfo.c` `tscreate.sqc` C++: `dbinfo.C` `dbrecov.sqC` `inauth.sqC` `ininfo.C` `tscreate.sqC`
db2CfgSet - Set Configuration Parameters	db2ApiDf	C: `dbinfo.c` `dbrecov.sqc` `ininfo.c` C++: `dbinfo.C` `dbrecov.sqC` `ininfo.C`
Database Directory Management		
sqlecadb - Catalog Database	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbcat.cbl`
sqleuncd - Uncatalog Database	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbcat.cbl`
sqlegdad - Catalog DCS Database	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dcscat.cbl`
sqlegdel - Uncatalog DCS Database	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dcscat.cbl`
sqledcgd - Change Database Comment	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbcmt.cbl`
sqledosd - Open Database Directory Scan	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbcat.cbl` `dbcmt.cbl`
sqledgne - Get Next Database Directory Entry	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbcat.cbl` `dbcmt.cbl`

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
sqledcls - Close Database Directory Scan	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbcat.cbl` `dbcmt.cbl`
sqlegdsc - Open DCS Directory Scan	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dcscat.cbl`
sqlegdgt - Get DCS Directory Entries	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dcscat.cbl`
sqlegdcl - Close DCS Directory Scan	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dcscat.cbl`
sqlegdge - Get DCS Directory Entry for Database	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dcscat.cbl`
Client/Server Directory Management		
sqlectnd - Catalog Node	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `nodecat.cbl`
sqleuncn - Uncatalog Node	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `nodecat.cbl`
sqlenops - Open Node Directory Scan	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `nodecat.cbl`
sqlengne - Get Next Node Directory Entry	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `nodecat.cbl`
sqlencls - Close Node Directory Scan	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `nodecat.cbl`
Network Support		
sqleregs - Register	sqlenv	n/a
sqledreg - Deregister	sqlenv	n/a
db2LdapRegister - LDAP Register Server	db2ApiDf	n/a
db2LdapUpdate - LDAP Update Server	db2ApiDf	n/a
db2LdapDeregister - LDAP Deregister Server	db2ApiDf	n/a
db2LdapCatalogNode - Catalog Node LDAP Entry	db2ApiDf	n/a
db2LdapUncatalogNode - Uncatalog Node LDAP Entry	db2ApiDf	n/a
db2LdapCatalogDatabase - Catalog Database LDAP Entry	db2ApiDf	n/a
db2LdapUncatalogDatabase - Uncatalog Database LDAP Entry	db2ApiDf	n/a

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
Recovery		
db2Backup - Backup database	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
sqlurcon - Reconcile	sqlutil	n/a
db2Restore - Restore database	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2Rollforward - Rollforward Database	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2HistoryOpenScan - Open History File Scan	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2HistoryGetEntry - Get Next History File Entry	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2HistoryCloseScan - Close History File Scan	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2Prune - Prune History File	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2HistoryUpdate - Update History File	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
Operational Utilities		
sqlefrce - Force Application	sqlenv	C: `dbconn.sqc` `dbsample.sqc` `instart.c` C++: `dbconn.sqC` `instart.C` COBOL: `dbstop.cbl`
db2Reorg - Reorganize	db2ApiDf	C: `tbreorg.sqc` C++: `tbreorg.sqC` COBOL: `dbstat.sqb`
db2Runstats - Runstats	db2ApiDf	C: `tbreorg.sqc` C++: `tbreorg.sqC` COBOL: `dbstat.sqb`
Database Monitoring		
db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot Output Buffer	db2ApiDf	n/a
db2MonitorSwitches - Get/Update Monitor Switches	db2ApiDf	C: `utilsnap.c` C++: `utilsnap.C`
db2GetSnapshot - Get Snapshot	db2ApiDf	C: `utilsnap.c` C++: `utilsnap.C`
db2ResetMonitor - Reset Monitor	db2ApiDf	n/a
db2ConvMonStream - Convert Monitor Stream	db2ApiDf	n/a
Health Monitoring		
db2AddContact - Add Contact	db2ApiDf	n/a
db2AddContactGroup - Add Contact Group	db2ApiDf	n/a

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
db2DropContact - Drop Contact	db2ApiDf	n/a
db2DropContactGroup - Drop Contact Group	db2ApiDf	n/a
db2GetAlertCfg - Get Alert Configuration	db2ApiDf	n/a
db2GetContactGroup - Get Contact Group	db2ApiDf	n/a
db2GetContactGroups - Get Contact Groups	db2ApiDf	n/a
db2GetContacts - Get Contacts	db2ApiDf	n/a
db2GetHealthNotificationList - Get Health Notification List	db2ApiDf	n/a
db2ResetAlertCfg - Reset Alert Configuration	db2ApiDf	n/a
db2UpdateAlertCfg - Update Alert Configuration	db2ApiDf	n/a
db2UpdateContact - Update Contact	db2ApiDf	n/a
db2UpdateContactGroup - Update Contact Group	db2ApiDf	n/a
db2UpdateHealthNotificationList - Update Health Notification List	db2ApiDf	n/a
Data Utilities		
sqluexpr - Export	sqlutil	C: `tbmove.sqc` C++: `tbmove.sqC` COBOL: `expsamp.sqb` `impexp.sqb` `tload.sqb`
sqluimpr - Import	sqlutil	C: `dtformat.sqc` `tbmove.sqc` C++: `tbmove.sqC` COBOL: `expsamp.sqb` `impexp.sqb`
db2Load - Load	db2ApiDf	C: `dtformat.sqc` `tbmove.sqc` C++: `tbmove.sqC`
db2LoadQuery - Load Query	db2ApiDf	C: `tbmove.sqc` C++: `tbmove.sqC` COBOL: `loadqry.sqb`
General Application Programming		
db2AutoConfig - Autoconfigure	db2AuCfg	C: `dbcfg.sqc` C++: `dbcfg.sqC`
db2AutoConfigFreeMemory - Free Autoconfigure Memory	db2AuCfg	C: `dbcfg.sqc` C++: `dbcfg.sqC`
sqlaintp - Get Error Message	sql	C: `dbcfg.sqc` `utilapi.c` C++: `dbcfg.sqC` `utilapi.C` COBOL: `checkerr.cbl`
sqlogstt - Get SQLSTATE Message	sql	C: `utilapi.c` C++: `utilapi.C` COBOL: `checkerr.cbl`

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
sqleisig - Install Signal Handler	sqlenv	COBOL: `dbcmt.cbl`
sqleintr - Interrupt	sqlenv	n/a
sqlgdref - Dereference Address	sqlutil	n/a
sqlgmcpy - Copy Memory	sqlutil	n/a
sqlefmem - Free Memory	sqlenv	C: `dbrecov.sqc` `tsinfo.sqc` C++: `dbrecov.sqC` `tsinfo.sqC` COBOL: `tabscont.sqb` `tabspace.sqb` `tspace.sqb`
sqlgaddr - Get Address	sqlutil	n/a
Application Preparation		
sqlaprep - Precompile Program	sql	C: `dbpkg.sqc` C++: `dbpkg.sqC`
sqlabndx - Bind	sql	C: `dbpkg.sqc` `dbsample.sqc` C++: `dbpkg.sqC`
sqlarbnd - Rebind	sql	C: `dbpkg.sqc` `dbsample.sqc` C++: `dbpkg.sqC` COBOL: `rebind.sqb`
Remote Server Utilities		
sqleatin - Attach	sqlenv	C: `inattach.c` `utilapi.c` C++: `inattach.C` `utilapi.C` COBOL: `dbinst.cbl`
sqleatcp - Attach and Change Password	sqlenv	C: `inattach.c` C++: `inattach.C` COBOL: `dbinst.cbl`
sqledtin - Detach	sqlenv	C: `inattach.c` `utilapi.c` C++: `inattach.C` `utilapi.C` COBOL: `dbinst.cbl`
Table Space Management		
sqlbtcq - Table Space Container Query	sqlutil	C: `dbrecov.sqc` `tsinfo.sqc` C++: `dbrecov.sqC` `tsinfo.sqC` COBOL: `tabscont.sqb` `tspace.sqb`
sqlbotcq - Open Table Space Container Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabscont.sqb` `tspace.sqb`
sqlbftcq - Fetch Table Space Container Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabscont.sqb` `tspace.sqb`
sqlbctcq - Close Table Space Container Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabscont.sqb` `tspace.sqb`
sqlbstsc - Set Table Space Containers	sqlutil	C: `dbrecov.sqc` C++: `dbrecov.sqC` COBOL: `tabscont.sqb` `tspace.sqb`
sqlbmtsq - Table Space Query	sqlutil	C: `dbrecov.sqc` `tsinfo.sqc` C++: `dbrecov.sqC` `tsinfo.sqC` COBOL: `tabspace.sqb` `tspace.sqb`
sqlbstpq - Single Table Space Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabspace.sqb` `tspace.sqb`
sqlbotsq - Open Table Space Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabspace.sqb` `tspace.sqb`

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
sqlbftpq - Fetch Table Space Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabspace.sqb` `tspace.sqb`
sqlbctsq - Close Table Space Query	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabspace.sqb` `tspace.sqb`
sqlbgtss - Get Table Space Statistics	sqlutil	C: `tsinfo.sqc` C++: `tsinfo.sqC` COBOL: `tabspace.sqb` `tspace.sqb`
sqluvqdp - Quiesce Table Spaces for Table	sqlutil	C: `tbmove.sqc` C++: `tbmove.sqC` COBOL: `tload.sqb`
Node Management		
sqleaddn - Add Node	sqlenv	n/a
sqledrpn - Drop Node Verify	sqlenv	n/a
Satellite		
db2GetSyncSession - Get Satellite Sync Session	db2ApiDf	n/a
db2QuerySatelliteProgress - Query Satellite Sync	db2ApiDf	n/a
db2SetSyncSession - Set Satellite Sync Session	db2ApiDf	n/a
db2SyncSatellite - Sync Satellite	db2ApiDf	n/a
db2SyncSatelliteStop - Stop Satellite Sync	db2ApiDf	n/a
db2SyncSatelliteTest - Test Satellite Sync	db2ApiDf	n/a
Database Partition Group Management		
sqludrdt - Redistribute Database Partition Group	sqlutil	n/a
Additional APIs		
sqluadau - Get Authorizations	sqlutil	C: `dbauth.sqc` `inauth.sqc` C++: `dbauth.sqC` `inauth.sqC`
sqlegins - Get Instance	sqlenv	C: `ininfo.c` C++: `ininfo.C` COBOL: `dbinst.cbl`
sqleqryc - Query Client	sqlenv	C: `cli_info.c` C++: `cli_info.C` COBOL: `client.cbl`
sqleqryi - Query Client Information	sqlenv	C: `cli_info.c` C++: `cli_info.C`
sqlesetc - Set Client	sqlenv	C: `cli_info.c` `dbcfg.sqc` `dbmcon.sqc` C++: `cli_info.C` `dbcfg.sqC` `dbmcon.sqC` COBOL: `client.cbl`

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
sqleseti - Set Client Information	sqlenv	C: `cli_info.c` C++: `cli_info.C`
sqlesact - Set Accounting String	sqlenv	C: `cli_info.c` C++: `cli_info.C` COBOL: `setact.cbl`
db2ReadLog - Asynchronous Read Log	db2ApiDf	C: `dbrecov.sqc` C++: `dbrecov.sqC`
db2ReadLogNoConn - Read Log Without a Database Connection	db2ApiDf	n/a
db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection	db2ApiDf	n/a
db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection	db2ApiDf	n/a
sqlugrpn - Get Row Partitioning Number	sqlutil	n/a
sqlugtpi - Get Table Partitioning Information	sqlutil	n/a
db2AdminMsgWrite - Administration Message Write	db2ApiDf	n/a
db2SetWriteForDB	db2ApiDf	n/a
db2ArchiveLog - Archive Active Log	db2ApiDf	n/a
db2DatabasePing - Ping Database	db2ApiDf	n/a
db2Inspect - Inspect database	db2ApiDf	n/a

Table 1. DB2 APIs, Include files, and Sample Programs (continued)

DB2 API	Include File	Sample Programs
Note: Include file extensions vary with programming language. C/C++ include files have a file extension of `.h`. COBOL include files have a file extension of `.cbl`. The include files can be found in the following directories: **C/C++ (UNIX):** `sqllib/include` **C/C++ (Windows):** `sqllib\include` **COBOL (UNIX):** `sqllib/include/cobol_a` `sqllib/include/cobol_i` `sqllib/include/cobol_mf` **COBOL (Windows):** `sqllib\include\cobol_a` `sqllib\include\cobol_i` `sqllib\include\cobol_mf`		

Table 2. C/C++ Sample Programs with DB2 APIs

Sample Program	Included APIs
`cli_info.c`, `cli_info.C`	• sqlesact - Set Accounting String • sqlesetc - Set Client • sqleseti - Set Client Information • sqleqryc - Query Client • sqleqryi - Query Client Information
`dbauth.sqc`, `dbauth.sqC`	• sqluadau - Get Authorizations
`dbcfg.sqc`, `dbcfg.sqC`	• db2AutoConfig - Autoconfigure • db2AutoConfigMemory - Free Autoconfigure Memory • sqlesetc - Set Client • sqlaintp - Get Error Message
`dbconn.sqc`, `dbconn.sqC`	• db2DatabaseRestart - Restart Database • sqlefrce - Force Application
`dbcreate.c`, `dbcreate.C`	• sqlecrea - Create Database • sqledrpd - Drop Database

Table 2. C/C++ Sample Programs with DB2 APIs (continued)

Sample Program	Included APIs
`dbinfo.c`, `dbinfo.C`	• db2CfgGet - Get Configuration • db2CfgSet - Set Configuration
`dbmcon.sqc`, `dbmcon.sqC`	• sqlesetc - Set Client
`dbmigrat.c`, `dbmigrat.C`	• sqlemgdb - Migrate Database
`dbpkg.sqc`, `dbpkg.sqC`	• sqlaprep - Precompile Program • sqlabndx - Bind • sqlarbnd - Rebind
`dbrecov.sqc`, `dbrecov.sqC`	• db2HistoryCloseScan - Close History File Scan • db2HistoryGetEntry - Get Next History File Entry • db2HistoryOpenScan - Open History File Scan • db2HistoryUpdate - Update History File • db2Prune - Prune History File • db2CfgGet - Get Configuration Parameters • db2CfgSet - Set Configuration Parameters • sqlbmtsq - Table Space Query • sqlbstsc - Set Table Space Containers • sqlbtcq - Table Space Container Query • sqlecrea - Create Database • sqledrpd - Drop Database • sqlefmem - Free Memory • db2Backup - Backup Database • db2Restore - Restore Database • db2ReadLog - Asynchronous Read Log • db2ReadLogNoConn - Read Log Without a Database Connection • db2Rollforward - Rollforward Database
`dbsample.sqc`	• db2DatabaseRestart - Restart Database • sqlecrea - Create Database • sqlefrce - Force Application • sqlabndx - Bind Package

Table 2. C/C++ Sample Programs with DB2 APIs (continued)

Sample Program	Included APIs
`dbthrds.sqc`, `dbthrds.sqC`	• sqleAttachToCtx - Attach to Context • sqleBeginCtx - Create and Attach to an Application Context • sqleDetachFromCtx - Detach From Context • sqleSetTypeCtx - Set Application Context Type
`dtformat.sqc`	• db2Load - Load • sqluimpr - Import
`inattach.c`, `inattach.C`	• sqleatcp - Attach and Change Password • sqleatin - Attach • sqledtin - Detach
`inauth.sqc`, `inauth.sqC`	• db2CfgGet - Get Configuration Parameters • sqluadau - Get Authorizations
`ininfo.c`, `ininfo.C`	• db2CfgGet - Get Configuration Parameters • db2CfgSet - Set Configuration Parameters • sqlegins - Get Instance • sqlectnd - Catalog Node • sqlenops - Open Node Directory Scan • sqlengne - Get Next Node Directory Entry • sqlencls - Close Node Directory Scan • sqleuncn - Uncatalog Node • sqlecadb - Catalog Database • sqledosd - Open Database Directory Scan • sqledgne - Get Next Database Directory Entry • sqledcgd - Change Database Comment • sqledcls - Close Database Directory Scan • sqleuncd - Uncatalog Database • sqlegdad - Catalog DCS Database • sqlegdsc - Open DCS Directory Scan • sqlegdge - Get DCS Directory Entry for Database • sqlegdgt - Get DCS Directory Entries • sqlegdcl - Close DCS Directory Scan • sqlegdel - Uncatalog DCS Database • sqlesdeg - Set Runtime Degree
`instart.c`, `instart.C`	• sqlefrce - Force Application • db2InstanceStart - Instance Start • db2InstanceStop - Instance Stop

Table 2. C/C++ Sample Programs with DB2 APIs (continued)

Sample Program	Included APIs
`tbmove.sqc`, `tbmove.sqC`	• sqluexpr - Export • sqluimpr - Import • sqluvqdp - Quiesce Table Spaces for Table • db2Load - Load • db2LoadQuery - Load Query
`tbreorg.sqc`, `tbreorg.sqC`	• db2Reorg - Reorganize • db2Runstats - Runstats
`tscreate.sqc`, `tscreate.sqC`	• db2CfgGet - Get Configuration Parameters
`tsinfo.sqc`, `tsinfo.sqC`	• sqlbstpq - Single Table Space Query • sqlbgtss - Get Table Space Statistics • sqlbmtsq - Table Space Query • sqlefmem - Free Memory • sqlbotsq - Open Table Space Query • sqlbftpq - Fetch Table Space Query • sqlbctsq - Close Table Space Query • sqlbtcq - Table Space Container Query • sqlbotcq - Open Table Space Container Query • sqlbftcq - Fetch Table Space Container Query • sqlbctcq - Close Table Space Container Query
`utilapi.c`, `utilapi.C`	• sqlaintp - Get Error Message • sqlogstt - Get SQLSTATE Message • sqleatin - Attach • sqledtin - Detach
`utilsnap.c`, `utilsnap.C`	• db2GetSnapshot - Get Snapshot • db2MonitorSwitches - Get/Update Monitor Switches

Table 3. COBOL Sample Programs with DB2 APIs

Sample Program	Included APIs
`checkerr.cbl`	• sqlaintp - Get Error Message • sqlogstt - Get SQLSTATE Message
`client.cbl`	• sqleqryc - Query Client • sqlesetc - Set Client

Table 3. COBOL Sample Programs with DB2 APIs (continued)

Sample Program	Included APIs
`db_udcs.cbl`	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database
`dbcat.cbl`	• sqlecadb - Catalog Database • sqledcls - Close Database Directory Scan • sqledgne - Get Next Database Directory Entry • sqledosd - Open Database Directory Scan • sqleuncd - Uncatalog Database
`dbcmt.cbl`	• sqledcgd - Change Database Comment • sqledcls - Close Database Directory Scan • sqledgne - Get Next Database Directory Entry • sqledosd - Open Database Directory Scan • sqleisig - Install Signal Handler
`dbinst.cbl`	• sqleatcp - Attach and Change Password • sqleatin - Attach • sqledtin - Detach • sqlegins - Get Instance
`dbstat.sqb`	• db2Reorg - Reorganize • db2Runstats - Runstats
`dcscat.cbl`	• sqlegdad - Catalog DCS Database • sqlegdcl - Close DCS Directory Scan • sqlegdel - Uncatalog DCS Database • sqlegdge - Get DCS Directory Entry for Database • sqlegdgt - Get DCS Directory Entries • sqlegdsc - Open DCS Directory Scan
`ebcdicdb.cbl`	• sqleatin - Attach • sqlecrea - Create Database • sqledrpd - Drop Database
`expsamp.sqb`	• sqluexpr - Export • sqluimpr - Import
`impexp.sqb`	• sqluexpr - Export • sqluimpr - Import
`loadqry.sqb`	• db2LoadQuery - Load Query

Table 3. COBOL Sample Programs with DB2 APIs (continued)

Sample Program	Included APIs
migrate.cbl	• sqlemgdb - Migrate Database
nodecat.cbl	• sqlectnd - Catalog Node • sqlencls - Close Node Directory Scan • sqlengne - Get Next Node Directory Entry • sqlenops - Open Node Directory Scan • sqleuncn - Uncatalog Node
rebind.sqb	• sqlarbnd - Rebind
tabscont.sqb	• sqlbctcq - Close Table Space Container Query • sqlbftcq - Fetch Table Space Container Query • sqlbotcq - Open Table Space Container Query • sqlbtcq - Table Space Container Query • sqlefmem - Free Memory
tabspace.sqb	• sqlbctsq - Close Table Space Query • sqlbftpq - Fetch Table Space Query • sqlbgtss - Get Table Space Statistics • sqlbmtsq - Table Space Query • sqlbotsq - Open Table Space Query • sqlbstpq - Single Table Space Query • sqlefmem - Free Memory
tload.sqb	• sqluexpr - Export • sqluvqdp - Quiesce Table Spaces for Table
tspace.sqb	• sqlbctcq - Close Table Space Container Query • sqlbctsq - Close Table Space Query • sqlbftcq - Fetch Table Space Container Query • sqlbftpq - Fetch Table Space Query • sqlbgtss - Get Table Space Statistics • sqlbmtsq - Table Space Query • sqlbotcq - Open Table Space Container Query • sqlbotsq - Open Table Space Query • sqlbstpq - Single Table Space Query • sqlbstsc - Set Table Space Containers • sqlbtcq - Table Space Container Query • sqlefmem - Free Memory
setact.cbl	• sqlesact - Set Accounting String

Related reference:

- "Include Files for C and C++" in the *Application Development Guide: Programming Client Applications*
- "Include Files for COBOL" in the *Application Development Guide: Programming Client Applications*
- "C/C++ Samples" in the *Application Development Guide: Building and Running Applications*
- "COBOL Samples" in the *Application Development Guide: Building and Running Applications*

How the API descriptions are organized

A short description of each API precedes some or all of the following subsections.

Scope:

The API's scope of operation within the instance. In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, `db2nodes.cfg`.

Authorization:

The authority required to successfully call the API.

Required connection:

One of the following: database, instance, none, or establishes a connection. Indicates whether the function requires a database connection, an instance attachment, or no connection to operate successfully. An explicit connection to the database or attachment to the instance may be required before a particular API can be called. APIs that require a database connection or an instance attachment can be executed either locally or remotely. Those that require neither cannot be executed remotely; when called at the client, they affect the client environment only.

API include file:

The name of the include file that contains the API prototype, and any necessary predefined constants and parameters.

Note: Include file extensions vary with programming language. C/C++ include files have a file extension of `.h`. COBOL include files have a file extension of `.cbl`. The include files can be found in the following directories:

C/C++ (UNIX):
`sqllib/include`

C/C++ (Windows):
`sqllib\include`

COBOL (UNIX):
`sqllib/include/cobol_a`

`sqllib/include/cobol_i`

`sqllib/include/cobol_mf`

COBOL (Windows):
`sqllib\include\cobol_a`

`sqllib\include\cobol_i`

`sqllib\include\cobol_mf`

C API syntax:

The C syntax of the API call.

Since Version 6, a new standard has been applied to the DB2 administrative APIs. Implementation of the new API definitions is being carried out in a staged manner. Following is a brief overview of the changes:

- The new API names contain the prefix "db2", followed by a meaningful mixed case string (for example, db2LoadQuery). Related APIs have names that allow them to be logically grouped. For example:
  ```
  db2HistoryCloseScan
  db2HistoryGetEntry
  db2HistoryOpenScan
  db2HistoryUpdate
  ```
- Generic APIs have names that contain the prefix "db2g", followed by a string that matches the C API name. Data structures used by generic APIs have names that also contain the prefix "db2g".
- The first parameter into the function (*versionNumber*) represents the version, release, or PTF level to which the code is to be compiled. This version number is used to specify the level of the structure that is passed in as the second parameter.
- The second parameter into the function is a void pointer to the primary interface structure for the API. Each element in the structure is either an

atomic type (for example, db2Long32) or a pointer. Each parameter name adheres to the following naming conventions:

```
piCamelCase  - pointer to input data
poCamelCase  - pointer to output data
pioCamelCase - pointer to input or output data
iCamelCase   - input data
ioCamelCase  - input/output data
oCamelCase   - output data
```

- The third parameter is a pointer to the SQLCA, and is mandatory.

Generic API syntax:

The syntax of the API call for the COBOL and FORTRAN programming languages.

Attention: Provide one extra byte for every character string passed to an API. Failure to do so may cause unexpected errors. This extra byte is modified by the database manager.

API parameters:

A description of each API parameter and its values. Predefined values are listed with the appropriate symbolics. Actual values for symbolics can be obtained from the appropriate language include files. COBOL programmers should substitute a hyphen (-) for the underscore (_) in all symbolics. For more information about parameter data types in each host language, see the sample programs.

Note: Applications calling database manager APIs must properly check for error conditions by examining return codes and the SQLCA structure. Most database manager APIs return a zero return code when successful. In general, a non-zero return code indicates that the secondary error handling mechanism, the SQLCA structure, may be corrupt. In this case, the called API is not executed. A possible cause for a corrupt SQLCA structure is passing an invalid address for the structure.

Error information is returned in the SQLCODE and SQLSTATE fields of the SQLCA structure, which is updated after most database manager API calls. Source files calling database manager APIs can provide one or more SQLCA structures; their names are arbitrary. An SQLCODE value of zero means successful execution (with possible SQLWARN warning conditions). A positive value means that the statement was successfully executed but with a warning, as with truncation of a host variable. A negative value means that an error condition occurred.

An additional field, SQLSTATE, contains a standardized error code that is consistent across other IBM database products, and across SQL92 compliant database managers. Use SQLSTATEs when concerned about portability, since SQLSTATEs are common across many database managers.

The SQLWARN field contains an array of warning indicators, even if SQLCODE is zero.

REXX API syntax:

The REXX syntax of the API call, where appropriate.

The SQLDB2 interface supports calling APIs from REXX. The SQLDB2 interface was created to provide support in REXX for new or previously unsupported APIs that do not have any output other than the SQLCA. Invoking a command through the SQLDB2 interface is syntactically the same as invoking the command through the command line processor (CLP), except that the token `call db2` is replaced by `CALL SQLDB2`. Using the `CALL SQLDB2` from REXX has the following advantages over calling the CLP directly:

- The compound REXX variable SQLCA is set
- By default, all CLP output messages are turned off.

REXX API parameters:

A description of each REXX API parameter and its values, where appropriate.

Usage notes:

Other information.

db2AddContact - Add Contact

Adds a contact to the contact list. Contacts are users to whom notification messages can be sent. Contacts can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2AddContact */
/* ... */
SQL_API_RC SQL_API_FN
db2AddContact (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2AddContactData
{
   char                       *piUserid;
   char                       *piPassword;
   char                       *piName;
   db2Uint32                  iType;
   char                       *piAddress;
   db2Uint32                  iMaxPageLength;
   char                       *piDescription;
} db2AddContactData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2AddContactData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piUserid;
Input. The user name.

piPassword
Input. The password for *piUserid*.

piName
Input. The contact name.

iType Input. Specifies the type of contact. Valid values are:
- DB2CONTACT_EMAIL
- DB2CONTACT_PAGE

piAddress
Input. The e-mail or pager address of the *iType* parameter.

iMaxPageLength
Input. The maximum message length for when *iType* is set to DB2CONTACT_PAGE.

piDescription
Input. User supplied description of the contact.

Related reference:

- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2DropContact - Drop Contact" on page 57
- "db2GetContacts - Get Contacts" on page 69
- "db2UpdateContact - Update Contact" on page 247

db2AddContactGroup - Add Contact Group

Adds a new contact group to the list of contact groups. A contact group contains a list of users to whom notification messages can be sent. Contact groups can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2AddContactGroup */
/* ... */
SQL_API_RC SQL_API_FN
db2AddContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2AddContactGroupData
{
   char                       *piUserid;
   char                       *piPassword;
   char                       *piGroupName;
   char                       *piDescription;
   db2Uint32                  iNumContacts;
   struct db2ContactTypeData *piContacts;
} db2AddContactGroupData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                  contactType;
   char                       *pName;
} db2ContactTypeData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2AddContactGroupData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piUserid
Input. The user name.

piPassword
Input. The password for *piUserid*.

piGroupName
Input. The name of the group to be retrieved.

piDescription
Input. The description of the group.

iNumContacts
Input. The number of *piContacts*.

piContacts
A pointer to the *db2ContactTypeData* structure.

contactType
Specifies the type of contact. Valid values are:
- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
The contact group name, or the contact name if *contactType* is set to DB2CONTACT_SINGLE.

Related reference:

- “SQLCA” on page 478
- “Location of Contact List configuration parameter - contact_host” in the *Administration Guide: Performance*
- “db2DropContactGroup - Drop Contact Group” on page 59
- “db2GetContactGroup - Get Contact Group” on page 66
- “db2GetContactGroups - Get Contact Groups” on page 67
- “db2UpdateContactGroup - Update Contact Group” on page 249

db2AdminMsgWrite - Administration Message Write

Provides a mechanism for users and Replication to write information to the `db2diag.log`, and the administration notification log.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2AdminMsgWrite */
/* ... */
SQL_API_RC  SQL_API_FN
db2AdminMsgWrite (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iMsgType;
  db2Uint32 iComponent;
  db2Uint32 iFunction;
  db2Uint32 iProbeID;
  char *piData_title;
  void *piData;
  db2Uint32 iDataLen;
  db2Uint32 iError_type;
} db2AdminMsgWriteStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2AdminMsgWriteStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iMsgType
Input. Specify the type of data to be logged. Valid values are `BINARY_MSG` for binary data, and `STRING_MSG` for string data.

iComponent
Input. Specify zero.

iFunction
Input. Specify zero.

iProbeID
Input. Specify the numeric probe point.

piData_title
Input. A pointer to the title string describing the data to be logged. Can be set to NULL if a title is not needed.

piData
Input. A pointer to the data to be logged. Can be set to NULL if data logging is not needed.

iDataLen
Input. The number of bytes of binary data to be used for logging if *iMsgType* is `BINARY_MSG`. Not used if *iMsgType* is `STRING_MSG`.

iError_type
Input. Valid values are:

```
DB2LOG_SEVERE_ERROR       (1) - Severe error has occurred
DB2LOG_ERROR              (2) - Error has occurred
DB2LOG_WARNING            (3) - Warning has occurred
DB2LOG_INFORMATION        (4) - Informational
```

Usage notes:

db2AdminMsgWrite - Administration Message Write

This API will log to the administration notification log only if the specified error type is less than or equal to the value of the *notifylevel* database manager configuration parameter. It will log to `db2diag.log` only if the specified error type is less than or equal to the value of the *diaglevel* database manager configuration parameter. However, all information written to the administration notification log is duplicated in the `db2diag.log` unless the *diaglevel* database manager configuration parameter is set to zero.

Related reference:

- "SQLCA" on page 478

db2ArchiveLog - Archive Active Log

Closes and truncates the active log file for a recoverable database. If user exit is enabled, issues an archive request.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

This API automatically establishes a connection to the specified database. If a connection to the specified database already exists, the API will return an error.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ArchiveLog */
/* ... */
SQL_API_RC SQL_API_FN
db2ArchiveLog (
  db2Uint32 version,
  void *pDB2ArchiveLogStruct,
  struct sqlca *pSqlca);

typedef struct
```

```
{
   char               *piDatabaseAlias;
   char               *piUserName;
   char               *piPassword;
   db2Uint16          iAllNodeFlag;
   db2Uint16          iNumNodes;
   SQL_PDB_NODE_TYPE  *piNodeList;
   db2Uint32          iOptions;
} db2ArchiveLogStruct
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gArchiveLog */
/* ... */
SQL_API_RC SQL_API_FN
db2gArchiveLog (
  db2Uint32 version,
  void *pDB2ArchiveLogStruct,
  struct sqlca *pSqlca);

typedef struct
{
   db2Uint32          iAliasLen;
   db2Uint32          iUserNameLen;
   db2Uint32          iPasswordLen;
   char               *piDatabaseAlias;
   char               *piUserName;
   char               *piPassword;
   db2Uint16          iAllNodeFlag;
   db2Uint16          iNumNodes;
   SQL_PDB_NODE_TYPE  *piNodeList;
   db2Uint32          iOptions;
} db2ArchiveLogStruct
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the variable passed in as the second parameter, *pDB2ArchiveLogStruct*.

pDB2ArchiveLogStruct
Input. A pointer to the *db2ArchiveLogStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

iUserNameLen
: Input. A 4-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is used.

iPasswordLen
: Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is used.

piDatabaseAlias
: Input. A string containing the database alias (as cataloged in the system database directory) of the database for which the active log is to be archived.

piUserName
: Input. A string containing the user name to be used when attempting a connection.

piPassword
: Input. A string containing the password to be used when attempting a connection.

iAllNodeFlag
: MPP only. Input. Flag indicating whether the operation should apply to all nodes listed in the `db2nodes.cfg` file. Valid values are:

  DB2ARCHIVELOG_NODE_LIST
  : Apply to nodes in a node list that is passed in *piNodeList*.

  DB2ARCHIVELOG_ALL_NODES
  : Apply to all nodes. *piNodeList* should be NULL. This is the default value.

  DB2ARCHIVELOG_ALL_EXCEPT
  : Apply to all nodes except those in the node list passed in *piNodeList*.

iNumNodes
: MPP only. Input. Specifies the number of nodes in the *piNodeList* array.

piNodeList
: MPP only. Input. A pointer to an array of node numbers against which to apply the archive log operation.

iOptions
: Input. Reserved for future use.

db2AutoConfig - Autoconfigure

Allows application programs to access the Performance Configuration wizard in the Control Center. Detailed information about this wizard is provided through the online help facility within the Control Center.

Authorization:

sysadm

Required connection:

Database

API include file:

db2AuCfg.h

C API syntax:

```
/* File: db2AuCfg.h */
/* API: db2AutoConfig */
/* ... */
SQL_API_RC_SQL_API_FN
db2AutoConfig(
  db2Uint32   db2VersionNumber,
  void *pAutoConfigInterface,
  struct sqlca *pSqlca);

typedef struct {
    db2int32  iProductID;
    char  iProductVersion[DB2_SG_PROD_VERSION_SIZE];
    char  iDbAlias[SQL_ALIAS_SZ];
    db2int32  iApply;
    db2AutoConfigInput  iParams;
    db2AutoConfigOutput  oResult;
} db2AutoConfigInterface;

typedef struct {
   db2int32  token;
   db2int32  value;
} db2AutoConfigElement;

typedef struct {
   db2Uint32  numElements;
   db2AutoConfigElement *pElements;
} db2AutoConfigArray;

typedef db2AutoConfigArray  db2AutoConfigInput;
typedef db2AutoConfigArray  db2AutoConfigDiags;

typedef struct {
   db2Uint32  numElements;
   struct sqlfupd *pConfigs;
   void *pDataArea;
} db2ConfigValues;

typedef struct {
   char *pName;
   db2int32  value;
} db2AutoConfigNameElement;

typedef struct {
   db2Uint32  numElements;
   db2AutoConfigElement *pElements;
} db2AutoConfigNameArray;

typedef db2AutoConfigNameArray db2BpValues;

typedef struct {
   db2ConfigValues  oOldDbValues;
   db2ConfigValues  oOldDbmValues;
   db2ConfigValues  oNewDbValues;
   db2ConfigValues  oNewDbmValues
   db2AutoConfigDiags  oDiagnostics;
   db2BpValues  oOldBpValues;
   db2BpValues  oNewBpValues;
} db2AutoConfigOutput;
/* ... */
```

API parameters:

db2VersionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pAutoConfigInterface*.

pAutoConfigInterface
: Input. A pointer to the *db2AutoConfigInterface* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iProductID
: Input. Specifies a unique product identifier. For valid Product ID values see the API include file `db2AuCfg.h`.

iProductVersion
: Input. A 16 byte string specifying the product version.

iDbAlias
: Input. A string specifying a database alias.

iApply
: Input. Updates the configuration automatically. For valid values see the API Include File `db2AuCfg.h`.

iParams
Input. Passes parameters into the wizard.

oResult
Output. Includes all results from the wizard.

token Specifies the configuration value for both the input parameters and the output diagnostics.

value Holds the data specified by the token.

numElements
The number of array elements.

pElements
A pointer to the element array.

db2AutoConfigDiags
Returns tokens and values for diagnostics and problem determination. The tokens identify the problems and the values state the recommendations when appropriate. For a list of tokens and values see the API Include File `db2AuCfg.h`.

pConfigs
A pointer to the SQLFUPD structure.

pDataArea
A pointer to the data area containing the values of the configuration.

pName
Output. The name of the output buffer pool.

value Holds the size (in pages) of the buffer pool specified in the name.

oOldDbValues
Output. If the *iApply* value is set to update the database configuration or all configurations, this value represents the database configuration value prior to using the wizard. Otherwise, this is the current value.

oOldDbmValues
Output. If the *iApply* value is set to update all configurations, this value represents the database manager configuration value prior to using the wizard. Otherwise, this is the current value.

oNewDbValues
Output. If the *iApply* value is set to update the database configuration or all configurations, this value represents the current database configuration value. Otherwise, this is the recommended value for the wizard.

oNewDbmValues
Output. If the *iApply* value is set to update all configurations, this

value represents the current database manager configuration value. Otherwise, this is the recommended value for the wizard.

oDiagnostics
Output. Includes diagnostics from the wizard.

oOldBpValues
Output. If the *iApply* value is set to update database configuration or all configurations, this value represents the buffer pool sizes in pages prior to using the wizard. Otherwise, this value is the current value.

oNewBpValues
Output. If the *iApply* value is set to update database configuration or all configurations, this value represents the current buffer pool sizes in pages. Otherwise, this is the recommended value for the wizard.

Usage notes:

To free the memory allocated by db2AutoConfig, call db2AutoConfigFreeMemory.

Related reference:
- "SQLCA" on page 478
- "SQLFUPD" on page 514
- "db2AutoConfigFreeMemory - Free Autoconfigure Memory" on page 30
- "db2CfgSet - Set Configuration Parameters" on page 42

Related samples:
- "dbcfg.sqc -- Configure database and database manager configuration parameters (C)"
- "dbcfg.sqC -- Configure database and database manager configuration parameters (C++)"

db2AutoConfigFreeMemory - Free Autoconfigure Memory

Frees the memory allocated by db2AutoConfig.

Authorization:

sysadm

Required connection:

Database

API include file:

db2AuCfg.h

C API syntax:

```
/* File: db2AuCfg.h */
/* API: db2AutoConfigFreeMemory */
/* ... */
SQL_API_RC_SQL_API_FN
db2AutoConfigFreeMemory(
    db2Uint32  db2VersionNumber,
    void *pAutoConfigInterface,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

db2VersionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pAutoConfigInterface*.

pAutoConfigInterface
Input. A pointer to the *db2AutoConfigInterface* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478
- "db2AutoConfig - Autoconfigure" on page 27

Related samples:

- "dbcfg.sqc -- Configure database and database manager configuration parameters (C)"
- "dbcfg.sqC -- Configure database and database manager configuration parameters (C++)"

db2Backup - Backup database

Creates a backup copy of a database or a table space.

Scope:

This API only affects the database partition on which it is executed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Database. This API automatically establishes a connection to the specified database.

The connection will be terminated upon the completion of the backup.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2Backup (
  db2Uint32     versionNumber,
  void         *pDB2BackupStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2BackupStruct
{
  char                         *piDBAlias;
  char                         oApplicationId[SQLU_APPLID_LEN+1];
  char                         oTimestamp[SQLU_TIME_STAMP_LEN+1];
  struct db2TablespaceStruct   *piTablespaceList;
  struct db2MediaListStruct    *piMediaList;
  char                         *piUsername;
  char                         *piPassword;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    oBackupSize;
  db2Uint32                    iCallerAction;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iParallelism;
  db2Uint32                    iOptions;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                         **tablespaces;
  db2Uint32                    numTablespaces;
} db2TablespaceStruct;
```

```
typedef SQL_STRUCTURE db2MediaListStruct
{
  char                         **locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2MediaListStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2gBackup (
  db2Uint32     versionNumber,
  void         *pDB2gBackupStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gBackupStruct
{
  char                         *piDBAlias;
  db2Uint32                    iDBAliasLen;
  char                         *poApplicationId;
  db2Uint32                    iApplicationIdLen;
  char                         *poTimestamp;
  db2Uint32                    iTimestampLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                         *piUsername;
  db2Uint32                    iUsernameLen;
  char                         *piPassword;
  db2Uint32                    iPasswordLen;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    oBackupSize;
  db2Uint32                    iCallerAction;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iParallelism;
  db2Uint32                    iOptions;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char               *tablespaces;
  db2Uint32                    numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char               *locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2gMediaListStruct;
```

```
typedef SQL_STRUCTURE db2Char
{
   char           *pioData;
   db2Uint32      iLength;
   db2Uint32      oLength;
} db2Char;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter *pDB2BackupStruct*.

pDB2BackupStruct
: Input. A pointer to the *db2BackupStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

piDBAlias
: Input. A string containing the database alias (as cataloged in the system database directory) of the database to back up.

iDBAliasLen
: Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

oApplicationId
: Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

poApplicationId
: Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil.h`). The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

iApplicationIdLen
: Input. A 4-byte unsigned integer representing the length in bytes of the *poApplicationId* buffer. Should be equal to SQLU_APPLID_LEN+1 (defined in `sqlutil.h`).

oTimestamp
: Output. The API will return the time stamp of the backup image

poTimestamp
: Output. Supply a buffer of length SQLU_TIME_STAMP_LEN+1 (defined in `sqlutil.h`). The API will return the time stamp of the backup image.

iTimestampLen
Input. A 4-byte unsigned integer representing the length in bytes of the *poTimestamp* buffer. Should be equal to SQLU_TIME_STAMP_LEN+1 (defined in `sqlutil.h`).

piTablespaceList
Input. List of table spaces to be backed up. Required for table space level backup only. Must be NULL for a database level backup. See structure DB2TablespaceStruct.

piMediaList
Input. This structure allows the caller to specify the destination for the backup operation. The information provided depends on the value of the *locationType* parameter. The valid values for *locationType* (defined in `sqlutil.h`) are:

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks, or diskettes).

SQLU_TSM_MEDIA
TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in the locations field.

SQLU_USER_EXIT
User exit. No additional input is required (only available when server is on OS/2).

For more information, see the *DB2MediaListStruct* structure .

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

iUsernameLen
Input. A 4-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

piVendorOptions
Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and code page is not checked for this data.

iVendorOptionsSize
Input. The length of the *piVendorOptions* field, which cannot exceed 65535 bytes.

oBackupSize
Output. Size of the backup image (in MB).

iCallerAction
Input. Specifies action to be taken. Valid values (defined in `db2ApiDf.h`) are:

DB2BACKUP_BACKUP
Start the backup.

DB2BACKUP_NOINTERRUPT
Start the backup. Specifies that the backup will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the backup have been mounted, and utility prompts are not desired.

DB2BACKUP_CONTINUE
Continue the backup after the user has performed some action requested by the utility (mount a new tape, for example).

DB2BACKUP_TERMINATE
Terminate the backup after the user has failed to perform some action requested by the utility.

DB2BACKUP_DEVICE_TERMINATE
Remove a particular device from the list of devices used by backup. When a particular medium is full, backup will return a warning to the caller (while continuing to process using the remaining devices). Call backup again with this caller action to remove the device which generated the warning from the list of devices being used.

DB2BACKUP_PARM_CHK
Used to validate parameters without performing a backup. This option does not terminate the database connection after the call returns. After successful return of this call, it is

expected that the user will issue a call with SQLUB_CONTINUE to proceed with the action.

DB2BACKUP_PARM_CHK_ONLY
Used to validate parameters without performing a backup. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

iBufferSize
Input. Backup buffer size in 4KB allocation units (pages). Minimum is 8 units. The default is 1024 units.

iNumBuffers
Input. Specifies number of backup buffers to be used. Minimum is 2. Maximum is limited by memory. Can specify 0 for the default value of 2.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024. The default is 1.

iOptions
Input. A bitmap of backup properties. The options are to be combined using the bitwise OR operator to produce a value for *iOptions*. Valid values (defined in `db2ApiDf.h`) are:

DB2BACKUP_OFFLINE
Offline gives an exclusive connection to the database.

DB2BACKUP_ONLINE
Online allows database access by other applications while the backup operation occurs.

Note: An online backup operation may appear to hang if users are holding locks on SMS LOB data.

DB2BACKUP_DB
Full database backup.

DB2BACKUP_TABLESPACE
Table space level backup. For a table space level backup, provide a list of table spaces in the *piTablespaceList* parameter.

DB2BACKUP_INCREMENTAL
Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DB2BACKUP_DELTA
Specifies a noncumulative (delta) backup image. A delta

backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

tablespaces
A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of *db2Char* structures.

numTablespaces
Number of entries in the *tablespaces* parameter.

locations
A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
The number of entries in the *locations* parameter.

locationType
A character indicated the media type. Valid values (defined in `sqlutil.h`.) are:

SQLU_LOCAL_MEDIA
Local devices (tapes, disks, diskettes, or named pipes).

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

pioData
A pointer to the character data buffer.

iLength
Input. The size of the *pioData* buffer.

oLength
Output. Reserved for future use.

Related reference:

- "sqlemgdb - Migrate Database" on page 368
- "db2Rollforward - Rollforward Database" on page 219
- "SQLCA" on page 478
- "db2Restore - Restore database" on page 208

Related samples:

- “dbrecov.sqc -- How to recover a database (C)”
- “dbrecov.sqC -- How to recover a database (C++)”

db2CfgGet - Get Configuration Parameters

Returns the values of individual entries in a specific database configuration file or a database manager configuration file.

Scope:

Information about a specific database configuration file is returned only for the database partition on which it is executed.

Authorization:

None

Required connection:

To obtain the current online value of a configuration parameter for a specific database configuration file, a connection to the database is required. To obtain the current online value of a configuration parameter for the database manager, an instance attachment is required. Otherwise, a connection to a database or an attachment to an instance is not required.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2CfgGet */
/* ... */
SQL_API_RC SQL_API_FN
db2CfgGet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32                          numItems;
   struct db2CfgParam                 *paramArray;
   db2Uint32                          flags;
   char                               *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
```

```
{
   db2Uint32                          token;
   char                               *ptrvalue;
   db2Uint32                          flags;
} db2CfgParam;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API db2gCfgGet */
/* ... */
SQL_API_RC SQL_API_FN
db2gCfgGet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32                          numItems;
   struct db2gCfgParam                *paramArray;
   db2Uint32                          flags;
   db2Uint32                          dbname_len;
   char                               *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32                          token;
   db2Uint32                          ptrvalue_len;
   char                               *ptrvalue;
   db2Uint32                          flags;
} db2gCfgParam;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
: Input. A pointer to the *db2Cfg* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

numItems
: Input. The number of configuration parameters in the *paramArray* array. Set this value to db2CfgMaxParam to specify the largest number of elements in the *paramArray*.

paramArray
Input. A pointer to the *db2CfgParam* structure.

flags (db2Cfg structure)
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf.h`) are:

db2CfgDatabase
Specifies to return the values in the database configuration file.

db2CfgDatabaseManager
Specifies to return the values in the database manager configuration file.

db2CfgImmediate
Returns the current values of the configuration parameters stored in memory.

db2CfgDelayed
Gets the values of the configuration parameters on disk. These do not become the current values in memory until the next database connection or instance attachment.

db2CfgGetDefaults
Returns the default values for the configuration parameter.

dbname_len
Input. The length in bytes of *dbname*.

dbname
Input. The database name.

token Input. The configuration parameter identifier.

ptrvalue_len
Input. The length in bytes of *ptrvalue*.

ptrvalue
Output. The configuration parameter value.

flags (db2CfgParam structure)
Input. Provides specific information for each parameter in a request. Valid values (defined in `db2ApiDf.h`) are:

db2CfgParamAutomatic
Indicates whether the parameter is set to *automatic*.

Related concepts:

- "Configuration parameter tuning" in the *Administration Guide: Performance*

Related tasks:

- "Configuring DB2 with configuration parameters" in the *Administration Guide: Performance*

Related reference:
- "SQLCA" on page 478
- "db2CfgSet - Set Configuration Parameters" on page 42

Related samples:
- "dbinfo.c -- Set and get information at the database level (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "inauth.sqc -- How to display authorities at instance level (C)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "tscreate.sqc -- How to create and drop buffer pools and table spaces (C)"
- "dbinfo.C -- Set and get information at the database level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "inauth.sqC -- How to display authorities at instance level (C++)"
- "ininfo.C -- Set and get information at the instance level (C++)"
- "tscreate.sqC -- How to create and drop buffer pools and table spaces (C++)"

db2CfgSet - Set Configuration Parameters

Modifies individual entries in a specific database configuration file or a database manager configuration file. A database configuration file resides on every node on which the database has been created.

Scope:

Modifications to the database configuration file affect the node on which it is executed.

Authorization:

For modifications to the database configuration file, one of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

For modifications to the database manager configuration file:
- *sysadm*

Required connection:

To make an online modification of a configuration parameter for a specific database, a connection to the database is required. To make an online modification of a configuration parameter for the database manager, an instance attachment is required. Otherwise a connection to a database or an attachment to an instance is not required.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2CfgSet */
/* ... */
SQL_API_RC SQL_API_FN
db2CfgSet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32                          numItems;
   struct db2CfgParam                 *paramArray;
   db2Uint32                          flags;
   char                               *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32                          token;
   char                               *ptrvalue;
   db2Uint32                          flags;
} db2CfgParam;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API db2gCfgGet */
/* ... */
SQL_API_RC SQL_API_FN
db2gCfgSet (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32                          numItems;
   struct db2gCfgParam                *paramArray;
   db2Uint32                          flags;
   db2Uint32                          dbname_len;
   char                               *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32                          token;
   db2Uint32                          ptrvalue_len;
   char                               *ptrvalue;
   db2Uint32                          flags;
} db2gCfgParam;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2Cfg* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

numItems
Input. The number of configuration parameters in the *paramArray* array.

paramArray
Input. A pointer to the *db2CfgParam* structure.

flags (db2Cfg structure)
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf.h`) are:

db2CfgDatabase
Specifies to return the values in the database configuration file.

db2CfgDatabaseManager
Specifies to return the values in the database manager configuration file.

db2CfgImmediate
Sets the current values of the configuration parameters in memory.

db2CfgDelayed
Sets the values of the configuration parameters on disk. These

do not become the current values in memory until the next database connection or instance attachment.

db2CfgReset
Resets the configuration parameters to the default values.

dbname_len
Input. The length in bytes of *dbname*.

dbname
Input. The database name.

token Input. The configuration parameter identifier.

ptrvalue_len
Input. The length in bytes of *ptrvalue*.

ptrvalue
Input. The configuration parameter value.

flags (db2CfgParam structure)
Input. Specifies the type of action to be taken for each parameter in a request. By default, this field should be set to zero. Valid values (defined in `db2ApiDf.h`) are:

db2CfgParamAutomatic
Sets the configuration parameter value to *automatic*. DB2 will automatically adjust this parameter to reflect the current resource requirements. Only parameters that support the automatic behavior can be set to *automatic*.

Related tasks:

- "Configuring DB2 with configuration parameters" in the *Administration Guide: Performance*

Related reference:

- "SQLCA" on page 478
- "Configuration parameters summary" in the *Administration Guide: Performance*
- "db2CfgGet - Get Configuration Parameters" on page 39

Related samples:

- "dbinfo.c -- Set and get information at the database level (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "dbinfo.C -- Set and get information at the database level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

- "ininfo.C -- Set and get information at the instance level (C++)"

db2ConvMonStream - Convert Monitor Stream

Converts the new, self-describing format for a single logical data element (for example, SQLM_ELM_DB2) to the corresponding pre-version 6 external monitor structure (for example, sqlm_db2). When upgrading API calls to use the post-version 5 stream, one must traverse the monitor data using the new stream format (for example, the user must find the SQLM_ELM_DB2 element). This portion of the stream can then be passed into the conversion API to get the associated pre-version 6 data.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ConvMonStream */
/* ... */
db2ConvMonStream (
  unsigned char version,
  db2ConvMonStreamData *data,
  struct sqlca *pSqlca);

typedef struct
{
  void *poTarget;
  sqlm_header_info *piSource;
  db2Uint32 iTargetType;
  db2Uint32 iTargetSize;
  db2Uint32 iSourceType
} db2ConvMonStreamData;
/* ... */
```

API parameters:

version
: Input. Specifies the version and release level of the structure passed in as the second parameter, *data*.

data Input. A pointer to the *db2ConvMonStreamData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

poTarget
Output. A pointer to the target monitor output structure (for example, sqlm_db2). A list of output types, and their corresponding input types, is given below.

piSource
Input. A pointer to the logical data element being converted (for example, SQLM_ELM_DB2). A list of output types, and their corresponding input types, is given below.

iTargetType
Input. The type of conversion being performed. Specify the value for the v5 type in `sqlmon.h` for instance SQLM_DB2_SS.

iTargetSize
Input. This parameter can usually be set to the size of the structure pointed to by *poTarget*; however, for elements that have usually been referenced by an offset value from the end of the structure (for example, statement text in *sqlm_stmt*), specify a buffer that is large enough to contain the sqlm_stmt statically-sized elements, as well as a statement of the largest size to be extracted; that is, `SQL_MAX_STMT_SZ` plus sizeof(sqlm_stmt).

iSourceType
Input. The type of source stream. Valid values are `SQLM_STREAM_SNAPSHOT` (snapshot stream), or `SQLM_STREAM_EVMON` (event monitor stream).

Usage notes:

Following is a list of supported convertible data elements:

Table 4. Supported convertible data elements: snapshot variables

Snapshot variable datastream type	Structure
SQLM_ELM_APPL	sqlm_appl
SQLM_ELM_APPL_INFO	sqlm_applinfo
SQLM_ELM_DB2	sqlm_db2
SQLM_ELM_FCM	sqlm_fcm
SQLM_ELM_FCM_NODE	sqlm_fcm_node
SQLM_ELM_DBASE	sqlm_dbase
SQLM_ELM_TABLE_LIST	sqlm_table_header

Table 4. Supported convertible data elements: snapshot variables (continued)

Snapshot variable datastream type	Structure
SQLM_ELM_TABLE	sqlm_table
SQLM_ELM_DB_LOCK_LIST	sqlm_dbase_lock
SQLM_ELM_APPL_LOCK_LIST	sqlm_appl_lock
SQLM_ELM_LOCK	sqlm_lock
SQLM_ELM_STMT	sqlm_stmt
SQLM_ELM_SUBSECTION	sqlm_subsection
SQLM_ELM_TABLESPACE_LIST	sqlm_tablespace_header
SQLM_ELM_TABLESPACE	sqlm_tablespace
SQLM_ELM_ROLLFORWARD	sqlm_rollfwd_info
SQLM_ELM_BUFFERPOOL	sqlm_bufferpool
SQLM_ELM_LOCK_WAIT	sqlm_lockwait
SQLM_ELM_DCS_APPL	sqlm_dcs_appl, sqlm_dcs_applid_info, sqlm_dcs_appl_snap_stats, sqlm_xid, sqlm_tpmon
SQLM_ELM_DCS_DBASE	sqlm_dcs_dbase
SQLM_ELM_DCS_APPL_INFO	sqlm_dcs_applid_info
SQLM_ELM_DCS_STMT	sqlm_dcs_stmt
SQLM_ELM_COLLECTED	sqlm_collected

Table 5. Supported convertible data elements: event monitor variables

Event monitor variable datastream type	Structure
SQLM_ELM_EVENT_DB	sqlm_db_event
SQLM_ELM_EVENT_CONN	sqlm_conn_event
SQLM_ELM_EVENT_TABLE	sqlm_table_event
SQLM_ELM_EVENT_STMT	sqlm_stmt_event
SQLM_ELM_EVENT_XACT	sqlm_xaction_event
SQLM_ELM_EVENT_DEADLOCK	sqlm_deadlock_event
SQLM_ELM_EVENT_DLCONN	sqlm_dlconn_event
SQLM_ELM_EVENT_TABLESPACE	sqlm_tablespace_event
SQLM_ELM_EVENT_DBHEADER	sqlm_dbheader_event
SQLM_ELM_EVENT_START	sqlm_evmon_start_event
SQLM_ELM_EVENT_CONNHEADER	sqlm_connheader_event
SQLM_ELM_EVENT_OVERFLOW	sqlm_overflow_event

Table 5. Supported convertible data elements: event monitor variables (continued)

Event monitor variable datastream type	Structure
SQLM_ELM_EVENT_BUFFERPOOL	sqlm_bufferpool_event
SQLM_ELM_EVENT_SUBSECTION	sqlm_subsection_event
SQLM_ELM_EVENT_LOG_HEADER	sqlm_event_log_header

The *sqlm_rollfwd_ts_info* structure is not converted; it only contains a table space name that can be accessed directly from the stream. The *sqlm_agent* structure is also not converted; it only contains the *pid* of the agent, which can also be accessed directly from the stream.

Related reference:

- "SQLCA" on page 478

db2DatabasePing - Ping Database

Tests the network response time of the underlying connectivity between a client and a database server. This API can be used by an application when a host database server is accessed via DB2 Connect either directly or through a gateway.

Authorization:

None

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabasePing */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabasePing (
  db2Uint32  versionNumber,
  void  *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DatabasePingStruct
```

```
{
char          iDbAlias[SQL_ALIAS_SZ + 1];
db2Uint16     iNumIterations;
db2Uint32    *poElapsedTime;
}
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gDatabasePing */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabasePing (
  db2Uint32     versionNumber,
  void         *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDatabasePingStruct
{
db2Uint16     iDbAliasLength;
char          iDbAlias[SQL_ALIAS_SZ];
db2Uint16     iNumIterations;
db2Uint32    *poElapsedTime;
}
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release of the DB2 Universal Database or DB2 Connect product that the application is using.

pParmStruct
: Input. A pointer to the *db2DatabasePingStruct* Structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iDbAliasLength
: Input. Length of the database alias name. Reserved for future use.

iDbAlias
: Input. Database alias name. Reserved for future use.

iNumIterations
: Input. Number of test request iterations. The value must be between 1 and 32767 inclusive.

poElapsedTime
: Output. A pointer to an array of 32-bit integers where the number of elements is equal to iNumIterations. Each element in the array will contain the elapsed time in microseconds for one test request iteration.

Note: The application is responsible for allocating the memory for this array prior to calling this API.

Usage notes:

This function can also be invoked using the PING command.

Related reference:

- "SQLCA" on page 478
- "PING Command" in the *Command Reference*

db2DatabaseQuiesce - Database Quiesce

Forces all users off the database, immediately rolls back all active transactions, and puts the database into quiesce mode. This API provides exclusive access to the database. During this quiesced period, system administration can be performed on the database. After administration is complete, you can unquiesce the database, using the db2DatabaseUnquiesce API. The db2DatabaseUnquiesce API allows other users to connect to the database, without having to shut down and perform another database start.

In this mode only groups or users with *QUIESCE CONNECT* authority and *sysadm*, *sysmaint*, or *sysctrl* will have access to the database and its objects.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DbQuiesceStruct
{
  char              *piDatabaseName;
  db2Uint32         iImmediate;
  db2Uint32         iForce;
} db2DbQuiesceStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gDatabaseQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabaseQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDbQuiesceStruct
{
  db2Uint32         iDatabaseNameLen;
  char              *piDatabaseName;
  db2Uint32         iImmediate;
  db2Uint32         iForce;
} db2gDbQuiesceStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2DbQuiesceStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iDatabaseNameLen
Input. Specifies the length in bytes of *piDatabaseName*.

piDatabaseName
Input. The database name.

iImmediate
Input. Reserved for future use.

iForce Input. Reserved for future use.

Related reference:

- "SQLCA" on page 478
- "db2DatabaseUnquiesce - Database Unquiesce" on page 53

db2DatabaseUnquiesce - Database Unquiesce

Restores user access to databases which have been quiesced for maintenance or other reasons. User access is restored without necessitating a shutdown and database restart.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DbUnquiesceStruct
{
             char          *piDatabaseName;
} db2DbUnquiesceStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gDatabaseunquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gDatabaseUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gDbUnquiesceStruct
{
            db2Uint32     iDatabaseNameLen;
            char          *piDatabaseName;
} db2gDbUnquiesceStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2DbUnquiesceStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iDatabaseNameLen
Input. Specifies the length in bytes of *piDatabaseName*.

piDatabaseName
Input. The database name.

Related reference:

- "SQLCA" on page 478
- "db2DatabaseQuiesce - Database Quiesce" on page 51

db2DatabaseRestart - Restart Database

Restarts a database that has been abnormally terminated and left in an inconsistent state. At the successful completion of this API, the application remains connected to the database if the user has CONNECT privilege.

Scope:

This API affects only the database partition server on which it is executed.

Authorization:

None

Required connection:

This API establishes a database connection.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseRestart */
/* ... */
SQL_API_RC SQL_API_FN
db2DatabaseRestart (
  db2Uint32 versionNumber;
  void *pParamStruct;
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseName;
  char *piUserId;
  char *piPassword;
  char *piTablespaceNames;
  int  *iOption;

} db2RestartDbStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DatabaseRestart */
/* ... */
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
    db2Uint32 versionNumber;
    void *pParamStruct;
    struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseName;
  char *piUserId;
  char *piPassword;
  char *piTablespaceNames;
  int  *iOption;

} db2RestartDbStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
Input. A pointer to the *db2RestartDbStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piDatabaseName
Input. A pointer to a string containing the alias of the database that is to be restarted.

piUserId
Input. A pointer to a string containing the user name of the application. May be NULL.

piPassword
Input. A pointer to a string containing a password for the specified user name (if any). May be NULL.

piTablespaceNames
Input. A pointer to a string containing a list of table space names to be dropped during the restart operation. May be NULL.

iOption
Input. Valid values are:

DB2_DB_SUSPEND_NONE
Performs normal crash recovery.

DB2_DB_RESUME_WRITE
Required to perform crash recovery on a database that has I/O writes suspended.

REXX API syntax:

```
RESTART DATABASE database_alias [USER username USING password]
```

REXX API parameters:

database_alias
Alias of the database to be restarted.

username
User name under which the database is to be restarted.

password
Password used to authenticate the user name.

Usage notes:

Call this API if an attempt to connect to a database returns an error message, indicating that the database must be restarted. This action occurs only if the previous session with this database terminated abnormally (due to power failure, for example).

At the completion of this API, a shared connection to the database is maintained if the user has CONNECT privilege, and an SQL warning is issued if any indoubt transactions exist. In this case, the database is still usable, but if the indoubt transactions are not resolved before the last connection to the database is dropped, another call to the API must be completed before the database can be used again.

In the case of circular logging, a database restart operation will fail if there is any problem with the table spaces, such as an I/O error, an unmounted file system, and so on. If losing such table spaces is not an issue, their names can be explicitly specified; this will put them into drop pending state, and the restart operation can complete successfully.

Related reference:

- "SQLCA" on page 478

Related samples:

- "dbconn.sqc -- How to connect to and disconnect from a database (C)"
- "dbconn.sqC -- How to connect to and disconnect from a database (C++)"

db2DropContact - Drop Contact

Removes a contact from the list of contacts. Contacts are users to whom notification messages can be sent.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

db2DropContact - Drop Contact

```
/* File: db2ApiDf.h */
/* API: db2DropContact */
/* ... */
SQL_API_RC SQL_API_FN
db2DropContact (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DropContactData
{
   char *piUserid;
   char *piPassword;
   char *piName;
} db2DropContactData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2DropContactData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piUserid;
Input. The user name.

piPassword
Input. The password for *piUserid*.

piName
Input. The name of the contact to be dropped.

Related reference:

- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContact - Add Contact" on page 18
- "db2GetContacts - Get Contacts" on page 69
- "db2UpdateContact - Update Contact" on page 247

db2DropContactGroup - Drop Contact Group

Removes a contact group from the list of contacts. A contact group contains a list of users to whom notification messages can be sent.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2DropContactGroup */
/* ... */
SQL_API_RC SQL_API_FN
db2DropContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DropContactData
{
   char *piUserid;
   char *piPassword;
   char *piName;
} db2DropContactData;
/* .. */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2DropContactData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piUserid
Input. The user name.

piPassword
Input. The password for *piUserid*.

piName
Input. The name of the contact group to be dropped.

Related reference:

- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContactGroup - Add Contact Group" on page 20
- "db2GetContactGroup - Get Contact Group" on page 66
- "db2GetContactGroups - Get Contact Groups" on page 67
- "db2UpdateContactGroup - Update Contact Group" on page 249

db2GetAlertCfg - Get Alert Configuration

Returns the alert configuration settings for the health indicators.

Authorization:

None

Required connection:

Instance. If there is not instance attachment, a default instance attachment is created.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API:  db2GetAlertCfg */
/* ... */
SQL_API_RC SQL_API_FN
db2GetAlertCfg(
  db2Uint32  versionNumber,
  void  *pParmStruct,
  struct sqlca *pSqlca );

typedef SQL_STRUCTURE db2GetAlertCfgData
{
   db2Uint32             iObjType;
   char                 *piObjName;
   db2Uint32            iDefault;
   char                *piDbname;
   db2Uint32            ioNumIndicators;
   db2GetAlertCfgInd   *pioIndicators;
} db2GetAlertCfgData;

typedef SQL_STRUCTURE db2GetAlertCfgInd
{
   db2Uint32              ioIndicatorID;
   db2int32               oAlarm;
   db2int32               oWarning;
   db2Uint32              oSensitivity;
   char                  *poFormula;
   db2Uint32              oActionEnabled;
   db2Uint32              oCheckThresholds;
   db2Uint32              oNumTaskActions;
   db2AlertTaskAction    *poTaskActions;
   db2Uint32              oNumScriptActions;
   db2AlertScriptAction  *poScriptActions;
} db2GetAlertCfgInd;

typedef SQL_STRUCTURE db2AlertTaskAction
{
   char                *pTaskname;
   db2Uint32            condition;
   char                *pUserId;
   char                *pPassword;
   char                *pHostName;
} db2AlertTaskAction;

typedef SQL_STRUCTURE db2AlertScriptAction
 {
   db2Uint32            scriptType;
   db2Uint32            condition;
   char                *pPathname;
   char                *pWorkingDir;
   char                *pCmdLineParms;
   char                 stmtTermChar;
   char                *pUserID;
   char                *pPassword;
   char                *pHostname;
} db2AlertScriptAction;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
: Input. A pointer to the *db2GetAlertCfgData* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iObjType
Input. Specifies the type of object for which configuration is requested. Valid values are:

- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES
- DB2ALERTCFG_OBJTYPE_TABLESPACES
- DB2ALERTCFG_OBJTYPE_TS_CONTAINERS
- DB2ALERTCFG_OBJTYPE_DATABASE
- DB2ALERTCFG_OBJTYPE_TABLESPACE
- DB2ALERTCFG_OBJTYPE_TS_CONTAINER

piObjName
Input. The name of the table space or table space container when the object type, *iObjType*, is set to DB2ALERTCFG_OBJTYPE_TABLESPACE or DB2ALERTCFG_OBJTYPE_TS_CONTAINER.

iDefault
Input. Indicates that the default installation configuration values are to be retrieved.

piDbname
Input. The alias name for the database for which configuration is requested when object type, *iObjType*, is DB2ALERTCFG_OBJTYPE_TS_CONTAINER, DB2ALERTCFG_OBJTYPE_TABLESPACE, and DB2ALERTCFG_OBJTYPE_DATABASE.

ioNumIndicators
This parameter can be used as either an input or output parameter.

Input. Indicates the number of *pioIndicators* submitted when requesting the settings for a subset of health indicators.

Output. Indicates the total number of health indicators returned by the API.

pioIndicators
A pointer to the *db2GetAlertCfgInd* structure. If it is set to NULL, all health indicators for that object will be returned.

ioIndicatorID
The health indicator (defined in `sqlmon.h`).

oAlarm
Output. The health indicator alarm threshold setting. This setting is valid for threshold-based health indicators only.

oWarning
: Output. The health indicator warning threshold setting. This setting is valid for threshold-based health indicators only.

oSensitivity
: Output. The period of time a health indicator's value must remain within a threshold zone before the associated alarm or warning condition is registered.

poFormula
: Output. A string representation of the formula used to compute the health indicator's value.

oActionEnabled
: Output. If TRUE, then any alert actions that are defined in *poTaskActions* or *poScriptActions* will be invoked if a threshold is breached. If FALSE, none of the defined actions will be invoked.

oCheckThresholds
: Output. If TRUE, the threshold breaches or state changes will be evaluated. If threshold breaches or states are not evaluated, then alerts will not be issued and alert actions will not be invoked regardless of whether *oActionEnabled* is TRUE.

oNumTaskActions
: Output. The number of task alert actions in the *pTaskAction* array.

poTaskActions
: A pointer to the *db2AlertTaskAction* structure.

oNumScriptActions
: Output. The number of script actions in the *poScriptActions* array.

poScriptActions
: A pointer to the *db2AlertScriptAction* structure.

pTaskname
: The name of the task.

condition
: The condition for which to run the action.

scriptType
: Specifies the type of script. Valid values are:

- DB2ALERTCFG_SCRIPTTYPE_DB2CMD
- DB2ALERTCFG_SCRIPTTYPE_OS

pPathname
: The absolute pathname of the script.

pWorkingDir
The absolute pathname of the directory in which the script is to be executed.

pCmdLineParms
The command line parameters to be passed to the script when it is invoked. Optional for DB2ALERTCFG_SCRIPTTYPE_OS only.

stmtTermChar
The character that is used in the script to terminate statements. Optional for DB2ALERTCFG_SCRIPTTYPE_DB2CMD only.

pUserID
The user account under which the script will be executed.

pPassword
The password for the user account *pUserId*.

pHostName
The host name on which to run the script. This applies for both task and script.

Script The hostname for where the script resides and will be run.

Task The hostname for where the scheduler resides.

Usage notes:

If *pioIndicators* is left NULL, all health indicators for that object will be returned. This parameter can be set to an array of *db2GetAlertCfgInd* structures with the *ioIndicatorID* set to the health indicator we desire to have the configuration for. When used in this manner, be sure to set *ioNumIndicators* to the input array length and to set all other fields in *db2GetAlertCfgInd* to 0 or NULL.

All of the memory under this pointer is allocated by the engine and must be freed with a db2GetAlertCfgFree call whenever db2GetAlertCfg returns with no error. See `db2ApiDf.h` for information about db2GetAlertCfgFree.

Related reference:
- “SQLCA” on page 478
- “db2ResetAlertCfg - Reset Alert Configuration” on page 202
- “db2UpdateAlertCfg - Update Alert Configuration” on page 242
- “Health Indicators” in the *System Monitor Guide and Reference*
- “db2GetAlertCfgFree - Free Get Alert Configuration Memory” on page 65

db2GetAlertCfgFree - Free Get Alert Configuration Memory

Frees the memory allocated by db2GetAlertCfg.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetAlertCfgFree */
/* ... */
SQL_API_RC_SQL_API_FN
db2GetAlertCfgFree (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2GetAlertCfgData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

Reference Text

Related reference:

- "SQLCA" on page 478
- "db2GetAlertCfg - Get Alert Configuration" on page 60
- "db2ResetAlertCfg - Reset Alert Configuration" on page 202
- "db2UpdateAlertCfg - Update Alert Configuration" on page 242

db2GetContactGroup - Get Contact Group

Returns the contacts included in a single contact group. Contacts are users to whom notification messages can be sent. Contacts can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetContactGroup */
/* ... */
SQL_API_RC SQL_API_FN
db2GetContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ContactGroupData
{
   char                        *pGroupName;
   char                        *pDescription;
   db2Uint32                   numContacts;
   struct db2ContactTypeData *pContacts;
} db2ContactGroupData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                   contactType;
   char                        *pName;
} db2ContactTypeData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2GetContactGroupData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

pGroupName
Input. The name of the group to be retrieved.

pDescription
The description of the group.

numContacts
The number of *pContacts*.

pContacts
A pointer to the *db2ContactTypeData* structure. The fields *pGroupName*, *pDescription*, *pContacts*, and *pContacts.pName* should be preallocated by the user with their respective maximum sizes. Call db2GetContactGroup with *numContacts*=0 and *pContacts*=NULL to have the required length for *pContacts* returned in *numContacts*.

contactType
Specifies the type of contact. Valid values are:

- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
The contact group name, or the contact name if *ioContactType* is set to DB2CONTACT_SINGLE.

Related reference:

- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContactGroup - Add Contact Group" on page 20
- "db2DropContactGroup - Drop Contact Group" on page 59
- "db2GetContactGroups - Get Contact Groups" on page 67
- "db2UpdateContactGroup - Update Contact Group" on page 249

db2GetContactGroups - Get Contact Groups

Returns the list of contact groups. Contacts are users to whom notification messages can be sent. Contact groups can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

db2GetContactGroups - Get Contact Groups

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetContactGroups */
/* ... */
SQL_API_RC SQL_API_FN
db2GetContactGroups (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetContactGroupsData
{
   db2Uint32                ioNumGroups;
   struct db2ContactGroupDesc *poGroups;
} db2GetContactGroupsData;

typedef SQL_STRUCTURE db2ContactGroupDesc
{
   char                     *poName;
   char                     *poDescription;
} db2ContactGroupDesc;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2GetContactGroupsData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

ioNumGroups
The number of groups. If *oNumGroups* = 0 and *poGroups* = NULL, it will contain the number of *db2ContactGroupDesc* structures needed in *poGroups*.

poGroups
Output. A pointer to the *db2ContactGroupDesc* structure.

poName
Output. The group name. This parameter should be preallocated by the caller with the respective maximum size.

poDescription
Output. The group description. This parameter should be preallocated by the caller with the respective maximum size.

Related reference:

- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContactGroup - Add Contact Group" on page 20
- "db2DropContactGroup - Drop Contact Group" on page 59
- "db2GetContactGroup - Get Contact Group" on page 66
- "db2UpdateContactGroup - Update Contact Group" on page 249

db2GetContacts - Get Contacts

Returns the list of contacts. Contacts are users to whom notification messages can be sent. Contacts can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetContacts */
/* ... */
SQL_API_RC SQL_API_FN
db2GetContacts (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetContactsData
{
   db2Uint32                ioNumContacts;
   struct db2ContactData    *poContacts;
} db2GetContactsData;

typedef SQL_STRUCTURE db2ContactData
{
   char                     *pName;
   db2Uint32                type;
   char                     *pAddress;
   db2Uint32                maxPageLength;
   char                     *pDescription;
} db2ContactData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2GetContactsData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

ioNumContacts
The number of *poContacts*.

poContacts
Output. A pointer to the *db2ContactData* structure. The fields *poContacts*, *pocontacts.pAddress*, *pocontacts.pDescription*, and *pocontacts.pName* should be preallocated by the user with their respective maximum sizes. Call db2GetContacts with *numContacts*=0 and *poContacts*=NULL to have the required length for *poContacts* returned in *numContacts*.

pName
The contact name.

type
Specifies the type of contact. Valid values are:
- DB2CONTACT_EMAIL
- DB2CONTACT_PAGE

pAddress
The address of the *type* parameter.

maxPageLength
The maximum message length for when *type* is set to DB2CONTACT_PAGE.

pDescription
User supplied description of the contact.

Related reference:
- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContact - Add Contact" on page 18
- "db2DropContact - Drop Contact" on page 57
- "db2UpdateContact - Update Contact" on page 247

db2GetHealthNotificationList - Get Health Notification List

Returns the list of contacts and/or contact groups that are notified about the health of an instance. A contact list consists of e-mail addresses or pager internet addresses of individuals who are to be notified when non-normal health conditions are present for an instance or any of its database objects.

Authorization:

None

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetHealthNotificationList */
/* ... */
SQL_API_RC SQL_API_FN
db2GetHealthNotificationList (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetHealthNotificationListData
```

```
{
   db2Uint32                ioNumContacts;
   struct db2ContactTypeData *poContacts;
} db2GetHealthNotificationListData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                contactType;
   char                     *pName;
} db2ContactTypeData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2GetHealthNotificationListData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

ioNumContacts
The number of contacts. If the API was called with a NULL *poContact*, then *ioNumContacts* will be set to the number of contacts the user should allocate to perform a successful call.

poContacts
Output. A pointer to the *db2ContactTypeData* structure.

contactType
Specifies the type of contact. Valid values are:
- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
The contact group name, or the contact name if *contactType* is set to DB2CONTACT_SINGLE. Set this value to a preallocated buffer of size DB2CONTACT_MAX_SZ.

Related reference:
- "SQLCA" on page 478
- "db2UpdateHealthNotificationList - Update Health Notification List" on page 251

db2GetSnapshot - Get Snapshot

Collects database manager monitor information and returns it to a user-allocated data buffer. The information returned represents a *snapshot* of the database manager operational status at the time the API was called.

Scope:

This API can return information for the database partition server on the instance, or all database partitions on the instance.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

To obtain a snapshot from a remote instance (or a different local instance), it is necessary to first attach to that instance.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetSnapshot */
/* ... */
SQL_API_RC SQL_API_FN
db2GetSnapshot (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetSnapshotData
{
  struct sqlma                *piSqlmaData;
  struct sqlm_collected       *poCollectedData;
  void                        *poBuffer;
  db2Uint32                   iVersion;
  db2Uint32                   iBufferSize;
  db2Uint32                iStoreResult;
  db2int32                 iNodeNumber;
  db2Uint32                *poOutputFormat;
  db2Uint32                iSnapshotClass;
} db2GetSnapshotData;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gGetSnapshot */
/* ... */
SQL_API_RC SQL_API_FN
db2gGetSnapshot (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gGetSnapshotData
{
  struct sqlma             *piSqlmaData;
  struct sqlm_collected    *poCollectedData;
  void                     *poBuffer;
  db2Uint32                iVersion;
  db2Uint32                iBufferSize;
  db2Uint32                iStoreResult;
  db2int32                 iNodeNumber;
  db2Uint32                *poOutputFormat;
  db2Uint32                iSnapshotClass;
} db2gGetSnapshotData;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter *pParmStruct*.

pParmStruct
: Input/Output. A pointer to the *db2GetSnapshotData* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

piSqlmaData
: Input. Pointer to the user-allocated *sqlma* (monitor area) structure. This structure specifies the type(s) of data to be collected.

poCollectedData
: Output. A pointer to the *sqlm_collected* structure into which the database monitor delivers summary statistics and the number of each type of data structure returned in the buffer area.

Note: This structure is only used for pre-Version 6 data streams. However, if a snapshot call is made to a back-level remote server, this structure must be passed in for results to be processed. It is therefore recommended that this parameter always be passed in.

poBuffer
Output. Pointer to the user-defined data area into which the snapshot information will be returned.

iVersion
Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- `SQLM_DBMON_VERSION1`
- `SQLM_DBMON_VERSION2`
- `SQLM_DBMON_VERSION5`
- `SQLM_DBMON_VERSION5_2`
- `SQLM_DBMON_VERSION6`
- `SQLM_DBMON_VERSION7`
- `SQLM_DBMON_VERSION8`

Note: If `SQLM_DBMON_VERSION1` is specified as the version, the APIs cannot be run remotely.

iBufferSize
Input. The length of the data buffer. Use db2GetSnapshotSize to estimate the size of this buffer. If the buffer is not large enough, a warning is returned, along with the information that will fit in the assigned buffer. It may be necessary to resize the buffer and call the API again.

iStoreResult
Input. An indicator set to `TRUE` or `FALSE`, depending on whether the snapshot results are to be stored at the DB2 server for viewing through SQL. This parameter should only be set to `TRUE` when the snapshot is being taken over a database connection, and when one of the snapshot types in the *sqlma* is `SQLMA_DYNAMIC_SQL`.

iNodeNumber
Input. The node where the request is to be sent. Based on this value, the request will be processed for the current node, all nodes or a user specified node. Valid values are:

- `SQLM_CURRENT_NODE`
- `SQLM_ALL_NODES`
- *`node value`*

Note: For standalone instances `SQLM_CURRENT_NODE` must be used.

poOutputFormat
The format of the stream returned by the server. It will be one of the following:

- `SQLM_STREAM_STATIC_FORMAT`
- `SQLM_STREAM_DYNAMIC_FORMAT`

iSnapshotClass
Input. The class qualifier for the snapshot. Valid values (defined in `sqlmon.h`) are:

- `SQLM_CLASS_DEFAULT` for a standard snapshot
- `SQLM_CLASS_HEALTH` for a health snapshot
- `SQLM_CLASS_HEALTH_WITH_DETAIL` for a health snapshot including additional details

Usage notes:

If an alias for a database residing at a different instance is specified, an error message is returned.

Related reference:

- “db2MonitorSwitches - Get/Update Monitor Switches” on page 177
- “db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot Output Buffer” on page 76
- “db2ResetMonitor - Reset Monitor” on page 205
- “SQLCA” on page 478
- “SQLM-COLLECTED” on page 515
- “SQLMA” on page 519
- “db2ConvMonStream - Convert Monitor Stream” on page 46

Related samples:

- “utilsnap.c -- Utilities for the snapshot monitor samples (C)”
- “utilsnap.C -- Utilities for the snapshot monitor samples (C++)”

db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot Output Buffer

Estimates the buffer size needed by ″db2GetSnapshot - Get Snapshot″.

Scope:

This API can either affect the database partition server on the instance, or all database partitions on the instance.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

To obtain information from a remote instance (or a different local instance), it is necessary to first attach to that instance. If an attachment does not exist, an implicit instance attachment is made to the node specified by the **DB2INSTANCE** environment variable.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetSnapshotSize */
/* ... */

SQL_API_RC SQL_API_FN
db2GetSnapshotSize (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2GetSnapshotSizeData
{
  struct sqlma               *piSqlmaData;
  sqluint32                  *poBufferSize;
  db2Uint32                  iVersion;
  db2int32                   iNodeNumber;
  db2Uint32                  iSnapshotClass;
} db2GetSnapshotSizeData;
/* ...*/
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gGetSnapshotSize */
/* ... */
SQL_API_RC SQL_API_FN
db2gGetSnapshotSize (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gGetSnapshotSizeData
{
  struct sqlma               *piSqlmaData;
  sqluint32                  *poBufferSize;
  db2Uint32                  iVersion;
  db2int32                   iNodeNumber;
  db2Uint32                  iSnapshotClass;
} db2gGetSnapshotSizeData;
/* ... */
```

API parameters:

version
: Input. Specifies the version and release level of the structure passed in as the second parameter *pParmStruct*.

pParmStruct
: Input. A pointer to the *db2GetSnapshotSizeStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

piSqlmaData
: Input. Pointer to the user-allocated *sqlma* (monitor area) structure. This structure specifies the type(s) of snapshot data to be collected, and can be reused as input to db2GetSnapshot.

poBufferSize
: Output. A pointer to the returned estimated buffer size needed by the GET SNAPSHOT API.

iVersion
: Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- `SQLM_DBMON_VERSION1`
- `SQLM_DBMON_VERSION2`
- `SQLM_DBMON_VERSION5`
- `SQLM_DBMON_VERSION5_2`
- `SQLM_DBMON_VERSION6`
- `SQLM_DBMON_VERSION7`

- `SQLM_DBMON_VERSION8`

Note: If `SQLM_DBMON_VERSION1` is specified as the version, the APIs cannot be run remotely.

iNodeNumber
Input. The database partition server where the request is to be sent. Based on this value, the request will be processed for the current database partition server, all database partition servers, or a user specified database partition server. Valid values are:

- `SQLM_CURRENT_NODE`
- `SQLM_ALL_NODES`
- *`node value`*

Note: For stand-alone instances `SQLM_CURRENT_NODE` must be used.

iSnapshotClass
Input. The class qualifier for the snapshot. Valid values (defined in `sqlmon.h`) are:

- `SQLM_CLASS_DEFAULT` for a standard snapshot
- `SQLM_CLASS_HEALTH` for a health snapshot
- `SQLM_CLASS_HEALTH_WITH_DETAIL` for a health snapshot including additional details

Usage notes:

This function generates a significant amount of overhead. Allocating and freeing memory dynamically for each db2GetSnapshot call is also expensive. If calling db2GetSnapshot repeatedly, for example, when sampling data over a period of time, it may be preferable to allocate a buffer of fixed size, rather than call db2GetSnapshotSize.

If the database system monitor finds no active databases or applications, it may return a buffer size of zero (if, for example, lock information related to a database that is not active is requested). Verify that the estimated buffer size returned by this API is non-zero before calling ″db2GetSnapshot - Get Snapshot″. If an error is returned by db2GetSnapshot because of insufficient buffer space to hold the output, call this API again to determine the new size requirements.

Related reference:

- “db2GetSnapshot - Get Snapshot” on page 73
- “db2MonitorSwitches - Get/Update Monitor Switches” on page 177
- “db2ResetMonitor - Reset Monitor” on page 205

- "SQLCA" on page 478
- "SQLMA" on page 519
- "Snapshot monitor logical data groups and data elements" in the *System Monitor Guide and Reference*
- "Event monitor logical data groups and data elements" in the *System Monitor Guide and Reference*

db2GetSyncSession - Get Satellite Sync Session

Gets the satellite's current synchronization session identifier.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GetSyncSession */
/* ... */
SQL_API_RC SQL_API_FN
  db2GetSyncSession (
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *poSyncSessionID;
} db2GetSyncSessionStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2GetSyncSessionStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

poSyncSessionID
Output. Specifies an identifier for the synchronization session that a satellite is currently using.

Related reference:

- "SQLCA" on page 478

db2HistoryCloseScan - Close History File Scan

Ends a history file scan and frees DB2 resources required for the scan. This API must be preceded by a successful call to db2HistoryOpenScan.

Authorization:

None

Required connection:

Instance. It is not necessary to call sqleatin before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
db2HistoryCloseScan (
  db2Uint32 version,
  void *piHandle,
  struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
db2GenHistoryCloseScan (
  db2Uint32 version,
   void *piHandle,
  struct sqlca *pSqlca);
/* ... */
```

db2HistoryCloseScan - Close History File Scan

API parameters:

version
Input. Specifies the version and release level of the second parameter, *piHandle*.

piHandle
Input. Specifies a pointer to the handle for scan access that was returned by db2HistoryOpenScan.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
CLOSE RECOVERY HISTORY FILE :scanid
```

REXX API parameters:

scanid Host variable containing the scan identifier returned from OPEN RECOVERY HISTORY FILE SCAN.

Usage notes:

For a detailed description of the use of the history file APIs, see db2HistoryOpenScan.

Related reference:

- "db2Prune - Prune History File" on page 180
- "db2HistoryUpdate - Update History File" on page 90
- "db2HistoryOpenScan - Open History File Scan" on page 86
- "db2HistoryGetEntry - Get Next History File Entry" on page 82
- "SQLCA" on page 478

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2HistoryGetEntry - Get Next History File Entry

Gets the next entry from the history file. This API must be preceded by a successful call to db2HistoryOpenScan.

Authorization:

None

Required connection:

Instance. It is not necessary to call sqleatin before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
db2HistoryGetEntry (
  db2Uint32 version,
  void *pDB2HistoryGetEntryStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData *pioHistData
} db2HistoryGetEntryStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
db2GenHistoryGetEntry (
  db2Uint32 version,
  void *pDB2GenHistoryGetEntryStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData *pioHistData
} db2GenHistoryGetEntryStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryGetEntryStruct*.

pDB2HistoryGetEntryStruct
Input. A pointer to the *db2HistoryGetEntryStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iHandle
Input. Contains the handle for scan access that was returned by db2HistoryOpenScan.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf`) are:

DB2HISTORY_GET_ENTRY
Get the next entry, but without any command data.

DB2HISTORY_GET_DDL
Get only the command data from the previous fetch.

DB2HISTORY_GET_ALL
Get the next entry, including all data.

pioHistData
Input. A pointer to the *db2HistData* structure.

REXX API syntax:

```
GET RECOVERY HISTORY FILE ENTRY :scanid [USING :value]
```

REXX API parameters:

scanid Host variable containing the scan identifier returned from OPEN RECOVERY HISTORY FILE SCAN.

value A compound REXX host variable into which the history file entry information is returned. In the following, XXX represents the host variable name:

XXX.0	Number of first level elements in the variable (always 15)
XXX.1	Number of table space elements
XXX.2	Number of used table space elements
XXX.3	OPERATION (type of operation performed)
XXX.4	OBJECT (granularity of the operation)
XXX.5	OBJECT_PART (time stamp and sequence number)
XXX.6	OPTYPE (qualifier of the operation)
XXX.7	DEVICE_TYPE (type of device used)
XXX.8	FIRST_LOG (earliest log ID)
XXX.9	LAST_LOG (current log ID)

XXX.10	BACKUP_ID (identifier for the backup)
XXX.11	SCHEMA (qualifier for the table name)
XXX.12	TABLE_NAME (name of the loaded table)
XXX.13.0	NUM_OF_TABLESPACES (number of table spaces involved in backup or restore)
XXX.13.1	Name of the first table space backed up/restored
XXX.13.2	Name of the second table space backed up/restored
XXX.13.3	and so on
XXX.14	LOCATION (where backup or copy is stored)
XXX.15	COMMENT (text to describe the entry).

Usage notes:

The records that are returned will have been selected using the values specified on the call to db2HistoryOpenScan.

For a detailed description of the use of the history file APIs, see db2HistoryOpenScan.

Related reference:
- “db2Prune - Prune History File” on page 180
- “db2HistoryUpdate - Update History File” on page 90
- “db2HistoryOpenScan - Open History File Scan” on page 86
- “db2HistoryCloseScan - Close History File Scan” on page 81
- “SQLCA” on page 478
- “db2HistData” on page 459

Related samples:
- “dbrecov.sqc -- How to recover a database (C)”
- “dbrecov.sqC -- How to recover a database (C++)”

db2HistoryOpenScan - Open History File Scan

Starts a history file scan.

Authorization:

None

Required connection:

Instance. If the database is cataloged as remote, call sqleatin before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
db2HistoryOpenScan (
  db2Uint32 version,
  void *pDB2HistoryOpenStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2HistoryOpenStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
db2GenHistoryOpenScan (
  db2Uint32 version,
  void *pDB2GenHistoryOpenStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2GenHistoryOpenStruct;
/* ... */
```

API parameters:

version
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryOpenStruct*.

pDB2HistoryOpenStruct
: Input. A pointer to the *db2HistoryOpenStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

piDatabaseAlias
: Input. A pointer to a string containing the database alias.

piTimestamp
: Input. A pointer to a string specifying the time stamp to be used for selecting records. Records whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL, or pointing to zero, prevents the filtering of entries using a time stamp.

piObjectName
: Input. A pointer to a string specifying the object name to be used for selecting records. The object may be a table or a table space. If it is a table, the fully qualified table name must be provided. Setting this parameter to NULL, or pointing to zero, prevents the filtering of entries using the object name.

oNumRows
: Output. Upon return from the API, this parameter contains the number of matching history file entries.

iCallerAction
: Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf`) are:

  DB2HISTORY_LIST_HISTORY
  : Lists all events that are currently logged in the history file.

  DB2HISTORY_LIST_BACKUP
  : Lists backup and restore operations.

  DB2HISTORY_LIST_ROLLFORWARD
  : Lists rollforward operations.

DB2HISTORY_LIST_DROPPED_TABLE
Lists dropped table records. The DDL field associated with an entry is not returned. To retrieve the DDL information for an entry, db2HistoryGetEntry must be called with a caller action of `DB2HISTORY_GET_DDL` immediately after the entry is fetched.

DB2HISTORY_LIST_LOAD
Lists load operations.

DB2HISTORY_LIST_CRT_TABLESPACE
Lists table space create and drop operations.

DB2HISTORY_LIST_REN_TABLESPACE
Lists table space renaming operations.

DB2HISTORY_LIST_ALT_TABLESPACE
Lists alter table space operations. The DDL field associated with an entry is not returned. To retrieve the DDL information for an entry, db2HistoryGetEntry must be called with a caller action of `DB2HISTORY_GET_DDL` immediately after the entry is fetched.

DB2HISTORY_LIST_REORG
Lists REORGANIZE TABLE operations. This value is not currently supported.

oHandle
Output. Upon return from the API, this parameter contains the handle for scan access. It is subsequently used in db2HistoryGetEntry, and db2HistoryCloseScan.

REXX API syntax:

```
OPEN [BACKUP] RECOVERY HISTORY FILE FOR database_alias
[OBJECT objname] [TIMESTAMP :timestamp]
USING :value
```

REXX API parameters:

database_alias
The alias of the database whose history file is to be listed.

objname
Specifies the object name to be used for selecting records. The object may be a table or a table space. If it is a table, the fully qualified table name must be provided. Setting this parameter to NULL prevents the filtering of entries using *objname*.

timestamp
Specifies the time stamp to be used for selecting records. Records

whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL prevents the filtering of entries using *timestamp*.

value A compound REXX host variable to which history file information is returned. In the following, XXX represents the host variable name.

XXX.0	Number of elements in the variable (always 2)
XXX.1	Identifier (handle) for future scan access
XXX.2	Number of matching history file entries.

Usage notes:

The combination of time stamp, object name and caller action can be used to filter records. Only records that pass all specified filters are returned.

The filtering effect of the object name depends on the value specified:
- Specifying a table will return records for load operations, because this is the only information for tables in the history file.
- Specifying a table space will return records for backup, restore, and load operations for the table space.

Note: To return records for tables, they must be specified as *schema.tablename*. Specifying *tablename* will only return records for table spaces.

A maximum of eight history file scans per process is permitted.

To list every entry in the history file, a typical application will perform the following steps:
1. Call db2HistoryOpenScan, which will return *oNumRows*.
2. Allocate an *db2HistData* structure with space for *n oTablespace* fields, where *n* is an arbitrary number.
3. Set the *iDB2NumTablespace* field of the *db2HistData* structure to *n*.
4. In a loop, perform the following:
   - Call db2HistoryGetEntry to fetch from the history file.
   - If db2HistoryGetEntry returns an SQLCODE of `SQL_RC_OK`, use the *sqld* field of the *db2HistData* structure to determine the number of table space entries returned.
   - If db2HistoryGetEntry returns an SQLCODE of `SQLUH_SQLUHINFO_VARS_WARNING`, not enough space has been allocated for all of the table spaces that DB2 is trying to return; free and reallocate the *db2HistData* structure with enough space for *oDB2UsedTablespace* table space entries, and set *iDB2NumTablespace* to *oDB2UsedTablespace*.

- If db2HistoryGetEntry returns an SQLCODE of `SQLE_RC_NOMORE`, all history file entries have been retrieved.
- Any other SQLCODE indicates a problem.

5. When all of the information has been fetched, call db2HistoryCloseScan to free the resources allocated by the call to db2HistoryOpenScan.

The macro SQLUHINFOSIZE(*n*) (defined in `sqlutil`) is provided to help determine how much memory is required for an *db2HistData* structure with space for *n oTablespace* fields.

Related reference:
- "db2Prune - Prune History File" on page 180
- "db2HistoryUpdate - Update History File" on page 90
- "db2HistoryGetEntry - Get Next History File Entry" on page 82
- "db2HistoryCloseScan - Close History File Scan" on page 81
- "SQLCA" on page 478

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2HistoryUpdate - Update History File

Updates the location, device type, or comment in a history file entry.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database. To update entries in the history file for a database other than the default database, a connection to the database must be established before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2HistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
db2HistoryUpdate (
  db2Uint32 version,
  void *pDB2HistoryUpdateStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piNewLocation,
  char *piNewDeviceType,
  char *piNewComment,
  db2Uint32 iEID
} db2HistoryUpdateStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
db2GenHistoryUpdate (
  db2Uint32 version,
  void *pDB2GenHistoryUpdateStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piNewLocation,
  char *piNewDeviceType,
  char *piNewComment,
  db2Uint32 iEID
} db2GenHistoryUpdateStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2HistoryUpdateStruct*.

pDB2HistoryUpdateStruct
Input. A pointer to the *db2HistoryUpdateStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piNewLocation
Input. A pointer to a string specifying a new location for the backup,

restore, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

piNewDeviceType
Input. A pointer to a string specifying a new device type for storing the backup, restore, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

piNewComment
Input. A pointer to a string specifying a new comment to describe the entry. Setting this parameter to NULL, or pointing to zero, leaves the comment unchanged.

iEID Input. A unique identifier that can be used to update a specific entry in the history file.

REXX API syntax:

```
UPDATE RECOVERY HISTORY USING :value
```

REXX API parameters:

value A compound REXX host variable containing information pertaining to the new location of a history file entry. In the following, XXX represents the host variable name:

XXX.0	Number of elements in the variable (must be between 1 and 4)
XXX.1	OBJECT_PART (time stamp with a sequence number from 001 to 999)
XXX.2	New location for the backup or copy image (this parameter is optional)
XXX.3	New device used to store the backup or copy image (this parameter is optional)
XXX.4	New comment (this parameter is optional).

Usage notes:

This is an update function, and all information prior to the change is replaced and cannot be recreated. These changes are not logged.

The primary purpose of the database history file is to record information, but the data contained in the history is used directly by automatic restore operations. During any restore where the AUTOMATIC option is specified, the history of backup images and their locations will be referenced and used by the restore utility to fulfill the automatic restore request. If the automatic restore function is to be used and backup images have been relocated since they were created, it is recommended that the database history record for those images be updated to reflect the current location. If the backup image location in the database history is not updated, automatic restore will not be able to locate the backup images, but manual restore commands can still be used successfully.

Related reference:

- "db2Rollforward - Rollforward Database" on page 219
- "db2Prune - Prune History File" on page 180
- "db2HistoryOpenScan - Open History File Scan" on page 86
- "db2HistoryGetEntry - Get Next History File Entry" on page 82
- "db2HistoryCloseScan - Close History File Scan" on page 81
- "SQLCA" on page 478
- "UPDATE HISTORY FILE Command" in the *Command Reference*
- "db2Backup - Backup database" on page 31

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2Inspect - Inspect database

Inspects the database for architectural integrity and checks the pages of the database for page consistency.

Scope:

In a single partition database, the scope is the single partition only. In a partitioned database environment, it is the collection of all logical partitions defined in `db2nodes.cfg`.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table

Required connection:

db2Inspect - Inspect database

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Inspect */
/* ... */
SQL_API_RC  SQL_API_FN
  db2Inspect (
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InspectStruct
{
  char                       *piTablespaceName;
  char                       *piTableName;
  char                       *piSchemaName;
  char                       *piResultsName;
  char                       *piDataFileName;
  SQL_PDB_NODE_TYPE          *piNodeList;
  db2Uint32                  iAction;
  db2int32                   iTablespaceID;
  db2int32                   iObjectID;
  db2Uint32                  iFirstPage;
  db2Uint32                  iNumberOfPages;
  db2Uint32                  iFormatType;
  db2Uint32                  iOptions;
  db2Uint32                  iBeginCheckOption;
  db2int32                   iLimitErrorReported;
  db2Uint16                  iObjectErrorState;
  db2Uint16                  iCatalogToTablespace;
  db2Uint16                  iKeepResultfile;
  db2Uint16                  iAllNodeFlag;
  db2Uint16                  iNumNodes;
  db2Uint16                  iLevelObjectData;
  db2Uint16                  iLevelObjectIndex;
  db2Uint16                  iLevelObjectLong;
  db2Uint16                  iLevelObjectLOB;
  db2Uint16                  iLevelObjectBlkMap;
  db2Uint16                  iLevelExtentMap;
} db2InspectStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInspect */
/* ... */
SQL_API_RC SQL_API_FN
  db2gInspect (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInspectStruct
{
  char                        *piTablespaceName;
  char                        *piTableName;
  char                        *piSchemaName;
  char                        *piResultsName;
  char                        *piDataFileName;
  SQL_PDB_NODE_TYPE           *piNodeList;
  db2Uint32                   iResultsNameLength;
  db2Uint32                   iDataFileNameLength;
  db2Uint32                   iTablespaceNameLength;
  db2Uint32                   iTableNameLength;
  db2Uint32                   iSchemaNameLength;
  db2Uint32                   iAction;
  db2int32                    iTablespaceID;
  db2int32                    iObjectID;
  db2Uint32                   iFirstPage;
  db2Uint32                   iNumberOfPages;
  db2Uint32                   iFormatType;
  db2Uint32                   iOptions;
  db2Uint32                   iBeginCheckOption;
  db2int32                    iLimitErrorReported;
  db2Uint16                   iObjectErrorState;
  db2Uint16                   iCatalogToTablespace;
  db2Uint16                   iKeepResultfile;
  db2Uint16                   iAllNodeFlag;
  db2Uint16                   iNumNodes;
  db2Uint16                   iLevelObjectData;
  db2Uint16                   iLevelObjectIndex;
  db2Uint16                   iLevelObjectLong;
  db2Uint16                   iLevelObjectLOB;
  db2Uint16                   iLevelObjectBlkMap;
  db2Uint16                   iLevelExtentMap;
} db2gInspectStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2InspectStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piTablespaceName
Input. A string containing the table space name. The table space must

be identified for operations on a table space. If the pointer is NULL, the table space ID value is used as input.

piTableName
Input. A string containing the table name. The table must be identified for operations on a table or a table object. If the pointer is NULL, the table space ID and table object ID values are used as input.

piSchemaName
Input. A string containing the schema name.

piResultsName
Input. A string containing the name for results output file. This input must be provided. The file will be written out to the diagnostic data directory path.

piDataFileName
Input. Reserved for future use. Must be set to NULL.

piNodeList
Input. A pointer to an array of partition numbers on which to perform the operation.

iResultsNameLength
Input. The string length of the results file name.

iDataFileNameLength
Input. The string length of the data output file name.

iTablespaceNameLength
Input. The string length of the table space name.

iTableNameLength
Input. The string length of the table name.

iSchemaNameLength
Input. The string length of the schema name.

iAction
Input. Specifies the inspect action. Valid values are:

DB2INSPECT_ACT_CHECK_DB
Inspect the entire database.

DB2INSPECT_ACT_CHECK_TABSPACE
Inspect a table space.

DB2INSPECT_ACT_CHECK_TABLE
Inspect a table.

DB2INSPECT_ACT_CHECK_RESTART
Restart the inspection.

DB2INSPECT_ACT_FORMAT_DATA
Format a data object page.

DB2INSPECT_ACT_FORMAT_INDEX
Format an index object page.

DB2INSPECT_ACT_FORMAT_PAGE
Format table space page.

DB2INSPECT_ACT_FORMAT_EXT_MAP
Format an extent map.

iTablespaceID
Input. Specifies the table space ID. If the table space must be identified, the table space ID value is used as input if the pointer to table space name is NULL.

iObjectID
Input. Specifies the object ID. If the table must be identified, the object ID value is used as input if the pointer to table name is NULL.

iFirstPage
Input. The page number of the first page to process. This must be specified for format data pages, index pages or table space pages.

iNumberOfPages
Input. The number of pages to process. This must be specified for format data pages, index pages or table space pages. To specify that processing should take place up to the last page, use the value DB2INSPECT_NUMPAGES_TO_THE_END.

iFormatType
Input. The type of formatting. This must be specified for format data pages or index pages. One of the following values must be specified:

DB2INSPECT_FORMAT_TYPE_DETAIL
Formatting in detail.

DB2INSPECT_FORMAT_TYPE_BRIEF
Formatting in brief.

DB2INSPECT_FORMAT_TYPE_HEX
Formatting in hexadecimal.

DB2INSPECT_FORMAT_TYPE_DEL
Formatting in delimited ASCII. Only applicable for data objects. The file name for the data file must also be provided in piDataFileName.

iOptions
Input. Options relevant to a particular *iAction*. Valid values are:

DB2INSPECT_OPTS_PAGE_LOGICAL
When formatting data pages or index pages, use this value if the first page number is logical page number.

DB2INSPECT_OPTS_PAGE_PHYSICAL
When formatting data pages or index pages, use this value if the first page number is physical page number.

iBeginCheckOption
Input. Option for check database or check table space operation to indicate where operation should begin. It must be set to zero to begin from the normal start. Values are:

DB2INSPECT_BEGIN_TSPID
Use this value for check database to begin with the table space specified by the table space ID field, the table space ID must be set.

DB2INSPECT_BEGIN_TSPID_OBJID
Use this value for check database to begin with the table specified by the table space ID and object ID field. To use this option, the table space ID and object ID must be set.

DB2INSPECT_BEGIN_OBJID
Use this value for check table space to begin with the table specified by the object ID field, the object ID must be set.

iLimitErrorReported
Input. Specifies the reporting limit of the number of pages in error for an object. Specify the number you want to use as the limit value or specify one the following values:

DB2INSPECT_LIMIT_ERROR_DEFAULT
Use this value to specify that the maximum number of pages in error to be reported is the extent size of the object.

DB2INSPECT_LIMIT_ERROR_ALL
Use this value to report all pages in error.

iObjectErrorState
Input. Specifies whether to scan objects in error state. Valid values are:

DB2INSPECT_ERROR_STATE_NORMAL
Process object only in normal state.

DB2INSPECT_ERROR_STATE_ALL
Process all objects, including objects in error state.

iCatalogToTablespace
Input. Inspects the consistency of the physical tables in the table space compared to the tables listed in the catalog. Valid values are:

DB2INSPECT_CAT_TO_TABSP_NONE
Specifies no consistency comparison of catalog to table space.

DB2INSPECT_CAT_TO_TABSP_YES
Specifies consistency comparison of catalog to table space.

iKeepResultfile
Input. Specifies result file retention. Valid values are:

DB2INSPECT_RESFILE_CLEANUP
If errors are reported, the result output file will be retained. Otherwise, the result file will be removed at the end of the operation.

DB2INSPECT_RESFILE_KEEP_ALWAYS
The result output file will be retained.

iAllNodeFlag
Input. Indicates whether the operation is to be applied to all nodes defined in `db2nodes.cfg`. Valid values are:

DB2_NODE_LIST
Apply to all nodes in a node list that is passed in *pNodeList*.

DB2_ALL_NODES
Apply to all nodes. *pNodeList* should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all nodes except those in a node list that is passed in *pNodeList*.

iNumNodes
Input. Specifies the number of nodes in the *pNodeList* array.

iLevelObjectData
Input. Specifies processing level for data object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectIndex
Input. Specifies processing level for index object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectLong
Input. Specifies processing level for long object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectLOB
Input. Specifies processing level for LOB object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectBlkMap
Input. Specifies processing level for block map object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelExtentMap
Input. Specifies processing level for extent map. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

Usage notes:

The online inspect processing will access database objects using isolation level uncommitted read. Commit processing will be done during the inspect processing. It is advisable to end the unit of work by issuing a COMMIT or ROLLBACK before starting the inspect operation.

The inspect check processing will write out unformatted inspection data results to the result file. The file will be written out to the diagnostic data directory path. If there are no errors found by the check processing, the result output file will be erased at the end of the inspect operation. If there are errors found by the check processing, the result output file will not be erased at the end of the inspect operation. To see the inspection details, format the inspection result output file with the `db2inspf` utility.

In a partitioned database environment, the extension of the result output file will correspond to the database partition number. The file is located in the database manager diagnostic data directory path.

A unique results output file name must be specified. If the result output file already exists, the operation will not be processed.

The processing of table spaces will process only the objects that reside in that table space.

Related reference:
- "SQLCA" on page 478

db2InstanceQuiesce - Instance Quiesce

Forces all users off the instance, immediately rolls back all active transaction, and puts the database into quiesce mode. This API provides exclusive access to the instance. During this quiesced period, system administration can be performed on the instance. After administration is complete, you can unquiesce the database using the db2DatabaseUnquiesce API. This API allows other users to connect to the database without having to shut down and perform another database start.

In this mode only groups or users with *QUIESCE CONNECT* authority and *sysadm*, *sysmaint*, or *sysctrl* will have access to the database and its objects.

Authorization:

One of the following:
- *sysadm*

- *sysctrl*

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2InstanceQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InsQuiesceStruct
{
             char          *piInstanceName;
             char          *piUserId;
             char          *piGroupId;
             db2Uint32     iImmediate;
             db2Uint32     iForce;
} db2InsQuiesceStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceQuiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gInstanceQuiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInsQuiesceStruct
{
             db2Uint32     iInstanceNameLen;
             char          *piInstanceName;
             db2Uint32     iUserIdLen;
             char          *piUserId;
             db2Uint32     iGroupIdLen;
             char          *piGroupId;
             db2Uint32     iImmediate;
             db2Uint32     iForce;
} db2gInsQuiesceStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct* .

pParmStruct
Input. A pointer to the *db2InsQuiesceStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iInstanceNameLen
Input. Specifies the length in bytes of *piInstanceName*.

piInstanceName
Input. The instance name.

iUserIdLen
Input. Specifies the length in bytes of *piUserID*.

piUserId
Input. The name of the a user who will be allowed access to the instance while it is quiesced.

iGroupIdLen
Input. Specifies the length in bytes of *piGroupId*.

piGroupId
Input. The name of a group that will be allowed access to the instance while the instance is quiesced.

iImmediate
Input. Reserved for future use.

iForce Input. Reserved for future use.

Related reference:
- "SQLCA" on page 478
- "db2InstanceUnquiesce - Instance Unquiesce" on page 114

db2InstanceStart - Instance Start

Starts a local or remote instance.

Scope:

In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, `db2nodes.cfg`.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2InstanceStart */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceStart (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InstanceStartStruct
{
             db2int8       iIsRemote;
             char          *piRemoteInstName;
             db2DasCommData *piCommData;
             db2StartOptionsStruct *piStartOpts;
} db2InstanceStartStruct;

typedef SQL_STRUCTURE db2DasCommData
{
             db2int8       iCommParam;
             char          *piNodeOrHostName;
             char          *piUserId;
             char          *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StartOptionsStruct
{
             db2Uint32     iIsProfile;
             char          *piProfile;
             db2Uint32     iIsNodeNum;
             db2NodeType   iNodeNum;
             db2Uint32     iOption;
             db2Uint32     iIsHostName;
             char          *piHostName;
             db2Uint32     iIsPort;
             db2PortType   iPort;
             db2Uint32     iIsNetName;
             char          *piNetName;
             db2Uint32     iTblspaceType;
             db2NodeType   iTblspaceNode;
             db2Uint32     iIsComputer;
             char          *piComputer;
             char          *piUserName;
             char          *piPassword;
             db2QuiesceStartStruct  iQuiesceOpts;
} db2StartOptionsStruct;

typedef SQL_STRUCTURE db2QuiesceStartStruct
{
             db2int8       iIsQRequested;
             char          *piQUsrName;
             char          *piQGrpName;
             db2int8       iIsQUsrGrpDef;
} db2QuiesceStartStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceStart */
SQL_API_RC SQL_API_FN
  db2gInstanceStart (
             db2Uint32 versionNumber,
             void *pParmStruct,
             struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInstanceStStruct
{
             db2int8       iIsRemote;
             db2Uint32     iRemoteInstLen;
             char          *piRemoteInstName;
             db2gDasCommData *piCommData;
             db2gStartOptionsStruct *piStartOpts;
} db2gInstanceStStruct;

typedef SQL&STRUCTURE db2gDasCommData
{
             db2int8       iCommParam;
             db2Uint32     iNodeOrHostNameLen;
             char          *piNodeOrHostName;
             db2Uint32     iUserIdLen;
             char          *piUserId;
             db2Uint32     iUserPwLen;
             char          *piUserPw;
} db2gDasCommData;

typedef SQL_STRUCTURE db2gStartOptionsStruct
{
             db2Uint32     iIsProfile;
             char          *piProfile;
             db2Uint32     iIsNodeNum;
             db2NodeType   iNodeNum;
             db2Uint32     iOption;
             db2Uint32     iIsHostName;
             char          *piHostName;
             db2Uint32     iIsPort;
             db2PortType   iPort;
             db2Uint32     iIsNetName;
             char          *piNetName;
             db2Uint32     iTblspaceType;
             db2NodeType   iTblspaceNode;
             db2Uint32     iIsComputer;
             char          *piComputer;
             char          *piUserName;
             char          *piPassword;
             db2gQuiesceStartStruct  iQuiesceOpts;
} db2gStartOptionsStruct;

typedef SQL_STRUCTURE db2gQuiesceStartStruct
{
             db2int8       iIsQRequested;
             db2Uint32     iQUsrNameLen;
             char          *piQUsrName;
             db2Uint32     iQGrpNameLen;
             char          *piQGrpName;
             db2int8       iIsQUsrGrpDef;
} db2gQuiesceStartStruct;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct* .

pParmStruct
: Input. A pointer to the *db2InstanceStartStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iIsRemote
: Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote start.

iRemoteInstLen
: Input. Specifies the length in bytes of *piRemoteInstName*.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the *db2DasCommData* structure.

piStartOpts
Input. A pointer to the *db2StartOptionsStruct* structure.

iCommParam
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote start.

iNodeOrHostNameLen
Input. Specifies the length in bytes of *piNodeOrHostName*.

piNodeOrHostName
Input. The database partition or hostname.

iUserIdLen
Input. Specifies the length in bytes of *piUserId*.

piUserId
Input. The user name.

iUserPwLen
Input. Specifies the length in bytes of *piUserPw*.

piUserPw
Input. The user password.

iIsProfile
Input. Indicates whether a profile is specified. If this field indicates that a profile is not specified, the file `db2profile` is used.

piProfile
Input. The name of the profile file to be executed at each node to define the DB2 environment (MPP only). This file is executed before the nodes are started. The default value is `db2profile`.

iIsNodeNum
Input. Indicates whether a node number is specified. If specified, the start command only affects the specified node.

iNodeNum
Input. The database partition number.

iOption
Input. Specifies an action. Valid values for *OPTION* (defined in `sqlenv.h`) are:

SQLE_NONE
Issue the normal db2start operation.

SQLE_ADDNODE
Issue the ADD NODE command.

SQLE_RESTART
Issue the RESTART DATABASE command.

SQLE_STANDALONE
Start the node in STANDALONE mode.

iIsHostName
Input. Indicates whether a host name is specified.

piHostName
Input. The system name.

iIsPort
Input. Indicates whether a port number is specified.

iPort Input. The port number.

iIsNetName
Input. Indicates whether a net name is specified.

piNetName
Input. The network name.

iTblspaceType
Input. Specifies the type of system temporary table space definitions to be used for the node being added. Valid values are:

SQLE_TABLESPACES_NONE
Do not create any system temporary table spaces.

SQLE_TABLESPACES_LIKE_NODE
The containers for the system temporary table spaces should be the same as those for the specified node.

SQLE_TABLESPACES_LIKE_CATALOG
The containers for the system temporary table spaces should be the same as those for the catalog node of each database.

iTblspaceNode
Input. Specifies the node number from which the system temporary table space definitions should be obtained. The node number must exist in the `db2nodes.cfg` file, and is only used if the *tblspace_type* field is set to `SQLE_TABLESPACES_LIKE_NODE`.

iIsComputer
Input. Indicates whether a computer name is specified. Valid on the Windows operating system only.

piComputer
Input. Computer name. Valid on the Windows operating system only.

piUserName
Input. Logon account user name. Valid on the Windows operating system only.

piPassword
Input. The password corresponding to the logon account user name.

iQuiesceOpts
Input. A pointer to the *db2QuiesceStartStruct* structure.

iIsQRequested
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if quiesce is requested.

iQUsrNameLen
Input. Specifies the length in bytes of *piQusrName*.

piQUsrName
Input. The quiesced username.

iQGrpNameLen
Input. Specifies the length in bytes of *piQGrpName.*

piQGrpName
Input. The quiesced group name.

iIsQUsrGrpDef
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if a quiesced user or quiesced group is defined.

Related reference:
- "SQLCA" on page 478
- "db2InstanceStop - Instance Stop" on page 109

Related samples:
- "instart.c -- Stop and start the current local instance (C)"
- "instart.C -- Stop and start the current local instance (C++)"

db2InstanceStop - Instance Stop

Stops the local or remote DB2 instance.

Scope:

In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, `db2nodes.cfg`.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2InstanceStop */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceStop (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InstanceStopStruct
{
             db2int8       iIsRemote;
             char          *piRemoteInstName;
             db2DasCommData *piCommData;
             db2StopOptionsStruct *piStopOpts;
} db2InstanceStopStruct;

typedef SQL_STRUCTURE db2DasCommData
{
             db2int8       iCommParam;
             char          *piNodeOrHostName;
             char          *piUserId;
             char          *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StopOptionsStruct
{
             db2Uint32     iIsProfile;
             char          *piProfile;
             db2Uint32     iIsNodeNum;
             db2NodeType   iNodeNum;
             db2Uint32     iStopOption;
             db2Uint32     iCallerac;
} db2StopOptionsStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceStop */
/* ... */
SQL_API_RC SQL_API_FN
db2gInstanceStop (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInstanceStopStruct
{
             db2int8       iIsRemote;
             db2Uint32     iRemoteInstLen;
             char          *piRemoteInstName;
             db2gDasCommData *piCommData;
             db2StopOptionsStruct *piStopOpts;
} db2gInstanceStopStruct;

typedef SQL_STRUCTURE db2gDasCommData
{
             db2int8       iCommParam;
             db2Uint32     iNodeOrHostNameLen;
             char          *piNodeOrHostName;
             db2Uint32     iUserIdLen;
             char          *piUserId;
             db2Uint32     iUserPwLen;
             char          *piUserPw;
} db2gDasCommData;

typedef SQL_STRUCTURE db2StopOptionsStruct
{
             db2Uint32     iIsProfile;
             char          *piProfile;
             db2Uint32     iIsNodeNum;
             db2NodeType   iNodeNum;
             db2Uint32     iStopOption;
             db2Uint32     iCallerac;
} db2StopOptionsStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct* .

pParmStruct
Input. A pointer to the *db2InstanceStopStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iIsRemote
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote start.

iRemoteInstLen
Input. Specifies the length in bytes of *piRemoteInstName*.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the *db2DasCommData* structure.

piStopOpts
Input. A pointer to the *db2StopOptionsStruct* structure.

iCommParam
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote stop.

iNodeOrHostNameLen
Input. Specifies the length in bytes of *piNodeOrHostName*.

piNodeOrHostName
Input. The database partition or hostname.

iUserIdLen
Input. Specifies the length in bytes of *piUserId*.

piUserId
Input. The user name.

iUserPwLen
Input. Specifies the length in bytes of *piUserPw*.

piUserPw
Input. The user password.

iIsRemote
Input. An indicator set to `TRUE` or `FALSE`. This parameter should be set to `TRUE` if this is a remote stop.

iRemoteInstLen
Input. Specifies the length in bytes of *piRemoteInstName*.

piRemoteInstName
Input. The remote instance name.

iIsProfile
Input. Indicates whether a profile is specified. If this field indicates that a profile is not specified, the file `db2profile` is used.

piProfile
Input. The name of the profile file that was executed at startup to

define the DB2 environment for those nodes that were started (MPP only). If a profile for the db2InstanceStart API was specified, the same profile must be specified here.

iIsNodeNum
Input. Indicates whether a node number is specified. If specified, the start command only affects the specified node.

iNodeNum
Input. The database partition number.

iStopOption
Input. Option. Valid values are:

SQLE_NONE
Issue the normal db2stop operation.

SQLE_FORCE
Issue the FORCE APPLICATION (ALL) command.

SQLE_DROP
Drop the node from the `db2nodes.cfg` file.

iCallerac
Input. This field is valid only for the SQLE_DROP value of the OPTION field. Valid values are:

SQLE_DROP
Initial call. This is the default value.

SQLE_CONTINUE
Subsequent call. Continue processing after a prompt.

SQLE_TERMINATE
Subsequent call. Terminate processing after a prompt.

Related reference:
- “SQLCA” on page 478
- “db2InstanceStart - Instance Start” on page 103

Related samples:
- “instart.c -- Stop and start the current local instance (C)”
- “instart.C -- Stop and start the current local instance (C++)”

db2InstanceUnquiesce - Instance Unquiesce

Unquiesce all databases in the instance.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2InstanceUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2InstanceUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2InsUnquiesceStruct
{
             char          *piInstanceName
} db2InsUnquiesceStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gInstanceUnquiesce */
/* ... */
SQL_API_RC SQL_API_FN
db2gInstanceUnquiesce (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gInsUnquiesceStruct
{
```

```
            db2Uint32     iInstanceNameLen;
            char          *piInstanceName;
} db2gInsUnquiesceStruct;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct* .

pParmStruct
: Input. A pointer to the *db2InsUnquiesceStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iInstanceNameLen
: Input. Specifies the length in bytes of *piInstanceName*.

piInstanceName
: Input. The instance name.

Related reference:

- "SQLCA" on page 478
- "db2InstanceQuiesce - Instance Quiesce" on page 101

db2LdapCatalogDatabase - Catalog Database LDAP Entry

Catalogs a database entry in LDAP (Lightweight Directory Access Protocol).

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapCatalogDatabase */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapCatalogDatabase(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
 char *piAlias;
 char *piDatabaseName;
 char *piComment
 char *piNodeName;
 char *piGWNodeName;
 char *piParameters;
 char *piARLibrary;
 unsigned short iAuthentication;
 char *piDCEPrincipalName;
 char *piBindDN;
 char *piPassword;
} db2LdapCatalogDatabaseStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
Input. A pointer to the *db2LdapCatalogDatabaseStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piAlias
Input. Specify an alias to be used as an alternate name for the database being cataloged. If an alias is not specified, the database manager uses the database name as the alias name.

piDatabaseName
Input. Specify the name of the database to catalog. This parameter is mandatory.

piComment
Input. Describes the DB2 server. Any comment that helps to describe the server registered in the network directory can be entered. Maximum length is 30 characters. A carriage return or a line feed character is not permitted.

piNodeName
: Input. Specify the node name of the database server on which the database resides. This parameter is required if the database resides on a remote database server.

piGWNodename
: Input. Specify the node name of the DB2 Connect gateway server. If the database server node type is DCS (reserved for host database servers), and the client does not have DB2 Connect installed, the client will connect to the DB2 Connect gateway server.

piParameters
: Input. Specify a parameter string that is to be passed to the application requester (AR). Authentication DCE is not supported.

piARLibrary
: Input. Specify the name of the application requester (AR) library.

iAuthentication
: Input. Specifying an authentication type can result in a performance benefit.

piDCEPrincipalName
: Input. Specify the fully qualified DCE principal name for the target server.

piBindDN
: Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to create and update the object in the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
: Input. Account password.

Usage notes:

A database may need to be manually registered or cataloged in LDAP if:

- The database server does not support LDAP. In this case, the administrator needs to manually register each database in LDAP to allow clients that support LDAP to access the database without having to catalog the database locally on each client machine.
- The application wants to use a different name to connect to the database. In this case, the administrator needs to catalog the database using a different alias name.
- During CREATE DATABASE IN LDAP, the database name already exists in LDAP. The database is still created on the local machine (and can be accessed by local applications), but the existing entry in LDAP will not be modified to reflect the new database. In this case, the administrator can:

– Remove the existing database entry from LDAP, and manually register the new database in LDAP.
– Register the new database in LDAP using a different alias name.

Related reference:

- "SQLCA" on page 478

db2LdapCatalogNode - Catalog Node LDAP Entry

Specifies an alternate name for the node entry in LDAP (Lightweight Directory Access Protocol), or a different protocol type for connecting to the database server.

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapCatalogNode */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapCatalogNode(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piAlias;
  char *piNodeName;
  char *piBindDN;
  char *piPassword;
} db2LdapCatalogNodeStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
Input. A pointer to the *db2LdapCatalogNodeStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piAlias
Input. Specify a new alias to be used as an alternate name for the node entry.

piNodeName
Input. Specify a node name that represents the DB2 server in LDAP.

piBindDN
Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to create and update the object in the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
Input. Account password.

Related reference:
- "SQLCA" on page 478

db2LdapDeregister - LDAP Deregister Server

Deregisters the DB2 server from LDAP (Lightweight Directory Access Protocol).

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapDeregister */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapDeregister (
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNodeName;
  char *piBindDN;
  char *piPassword;
} db2LdapDeregisterStruct;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
: Input. A pointer to the *db2LdapDeregisterStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

piNodeName
: Input. Specify a short name that represents the DB2 server in LDAP.

piBindDN
: Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to delete the object from the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
: Input. Account password.

Related reference:

- "SQLCA" on page 478

db2LdapRegister - LDAP Register Server

Registers the DB2 server in LDAP (Lightweight Directory Access Protocol).

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapRegister */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapRegister (
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piNodeName;
  char *piComputer;
  char *piInstance;
  unsigned short iNodeType;
  db2LdapProtocolInfo iProtocol;
  char *piComment;
  char *piBindDN;
  char *piPassword;
} db2LdapRegisterStruct;

typedef struct
{
  char iType;
  char *piHostName;
  char *piServiceName;
  char *piNetbiosName;
  char *piNetworkId;
  char *piPartnerLU;
  char *piTPName;
  char *piMode;
  unsigned short iSecurityType;
  char *piLanAdapterAddress;
  char *piChangePasswordLU;
  char *piIpxAddress;
} db2LdapProtocolInfo;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
Input. A pointer to the *db2LdapRegisterStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piNodeName
Input. Specify a short name (less than 8 characters) that represents the DB2 server in LDAP.

piComputer
Input. Specify the name of the computer system on which the DB2 server resides. The computer name value must be the same as the value specified when adding the server machine to LDAP. On Windows NT, this is the NT computer name. On UNIX based systems, this is the TCP/IP host name. On OS/2, this is the value specified for the **DB2SYSTEM** registry variable. Specify NULL to register the DB2 server on the local computer.

piInstance
Input. Specify the instance name of the DB2 server. The instance name must be specified if the computer name is specified to register a remote server. Specify NULL to register the current instance (as defined by the **DB2SYSTEM** environment variable).

iNodeType
Input. Specify the node type for the database server. Valid values are:

```
SQLF_NT_SERVER
SQLF_NT_MPP
SQLF_NT_DCS
```

iProtocol
Input. Specify the protocol information in the *db2LdapProtocolInfo* structure.

piComment
Input. Describes the DB2 server. Any comment that helps to describe

the server registered in the network directory can be entered. Maximum length is 30 characters. A carriage return or a line feed character is not permitted.

piBindDN
Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to create and update the object in the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
Input. Account password.

iType Input. Specify the protocol type that this server supports. If the server supports more than one protocol, multiple registrations (each with a different node name and protocol type) are required. Valid values are:

```
SQL_PROTOCOL_APPN   - For APPC/APPN support
SQL_PROTOCOL_NETB   - For NetBIOS support
SQL_PROTOCOL_TCPIP  - For TCP/IP support
SQL_PROTOCOL_SOCKS  - For TCP/IP with socket security
SQL_PROTOCOL_IPXSPX - For IPX/SPX support
SQL_PROTOCOL_NPIPE  - For Windows NT Named Pipe support
```

piHostName
Input. Specify the TCP/IP host name or the IP address.

piServiceName
Input. Specify the TCP/IP service name or port number.

piNetbiosName
Input. Specify the NetBIOS workstation name. The NetBIOS name must be specified for NetBIOS support.

piNetworkID
Input. Specify the network ID. The network ID must be specified for APPC/APPN support.

piPartnerLU
Input. Specify the partner LU name for the DB2 server machine. The partner LU must be specified for APPC/APPN support.

piTPName
Input. Specify the transaction program name. The transaction program name must be specified for APPC/APPN support.

piMode
Input. Specify the mode name. The mode must be specified for APPC/APPN support.

iSecurityType
Input. Specify the APPC security level. Valid values are:

```
SQL_CPIC_SECURITY_NONE (default)
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM
```

piLanAdapterAddress
Input. Specify the network adapter address. This parameter is only required for APPC support. For APPN, this parameter can be set to NULL.

piChangePasswordLU
Input. Specify the name of the partner LU to use when changing the password for the host database server.

piIpxAddress
Input. Specify the complete IPX address. The IPX address must be specified for IPX/SPX support.

Usage notes:

Register the DB2 server once for each protocol that the server supports each time specifying a unique node name.

If any protocol configuration parameter is specified when registering a DB2 server locally, it will override the value specified in the database manager configuration file.

Only a remote DB2 server can be registered in LDAP. The computer name and the instance name of the remote server must be specified, along with the protocol communication for the remote server.

When registering a host database server, a value of `SQLF_NT_DCS` must be specified for the *iNodeType* parameter.

Related reference:
- "SQLCA" on page 478

db2LdapUncatalogDatabase - Uncatalog Database LDAP Entry

Removes a database entry from LDAP (Lightweight Directory Access Protocol).

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapUncatalogDatabase */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUncatalogDatabase(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piAlias[SQL_ALIAS_SZ];
  char *piBindDN;
  char *piPassword;
} db2LdapUncatalogDatabaseStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
Input. A pointer to the *db2LdapUncatalogDatabaseStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piAlias
Input. Specify an alias name for the database entry. This parameter is mandatory.

piBindDN
Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to delete the object from the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
Input. Account password.

Related reference:
- "SQLCA" on page 478

db2LdapUncatalogNode - Uncatalog Node LDAP Entry

Removes a node entry from LDAP (Lightweight Directory Access Protocol).

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapUncatalogNode */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUncatalogNode(
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piAlias;
  char *piBindDN;
  char *piPassword;
} db2LdapUncatalogNodeStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParamStruct
Input. A pointer to the *db2LdapUncatalogNodeStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piAlias
Input. Specify the alias of the node to uncatalog from LDAP.

piBindDN
Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to delete the object from the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
Input. Account password.

Related reference:

- "SQLCA" on page 478

db2LdapUpdate - LDAP Update Server

Updates the communication protocol information for the DB2 server in LDAP (Lightweight Directory Access Protocol).

This API is available on supported Windows platforms only.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LdapUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2LdapUpdate (
    sqlint32 versionNumber,
    void *pParamStruct,
    struct sqlca *pSqlca);

typedef struct
```

```
{
  char *piNodeName;
  char *piComment;
  unsigned short iNodeType;
  db2LdapProtocolInfo iProtocol;
  char *piBindDN;
  char *piPassword;
} db2LdapUpdateStruct;

typedef struct
{
  char iType;
  char *piHostName;
  char *piServiceName;
  char *piNetbiosName;
  char *piNetworkId;
  char *piPartnerLU;
  char *piTPName;
  char *piMode;
  unsigned short iSecurityType;
  char *piLanAdapterAddress;
  char *piChangePasswordLU;
  char *piIpxAddress;
} db2LdapProtocolInfo;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParamStruct*.

pParamStruct
Input. A pointer to the *db2LdapUpdateStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piNodeName
Input. Specify the node name that represents the DB2 server in LDAP.

piComment
Input. Specify a new description for the DB2 server. Maximum length is 30 characters. A carriage return or a line feed character is not permitted.

iNodeType
Input. Specify a new node type. Valid values are:

```
SQLF_NT_SERVER
SQLF_NT_MPP
SQLF_NT_DCS
SQL_PARM_UNCHANGE
```

iProtocol
Input. Specify the updated protocol information in the *db2LdapProtocolInfo* structure.

piBindDN
Input. Specify the user's LDAP distinguished name (DN). The LDAP user DN must have sufficient authority to create and update the object in the LDAP directory. If the user's LDAP DN is not specified, the credentials of the current logon user will be used.

piPassword
Input. Account password.

iType Input. Specify the protocol type that this server supports. Valid values are:

```
SQL_PROTOCOL_APPN   - For APPC/APPN support
SQL_PROTOCOL_NETB   - For NetBIOS support
SQL_PROTOCOL_TCPIP  - For TCP/IP support
SQL_PROTOCOL_SOCKS  - For TCP/IP with socket security
SQL_PROTOCOL_IPXSPX - For IPX/SPX support
SQL_PROTOCOL_NPIPE  - For Windows NT Named Pipe support
```

piHostName
Input. Specify a new TCP/IP host name or IP address.

piServiceName
Input. Specify a new TCP/IP service name or port number.

piNetbiosName
Input. Specify a new NetBIOS workstation name.

piNetworkID
Input. Specify a new network ID.

piPartnerLU
Input. Specify a new partner LU name for the DB2 server machine.

piTPName
Input. Specify a new transaction program name.

piMode
Input. Specify a new mode name.

iSecurityType
Input. Specify a new security level. Valid values are:

```
SQL_CPIC_SECURITY_NONE
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM
SQL_PARM_UNCHANGE
```

piLanAdapterAddress
Input. Specify a new network adapter address.

piChangePasswordLU
Input. Specify a new name of the partner LU to use when changing the password for the host database server.

piIpxAddress
Input. Specify a new IPX address.

Related reference:

- "SQLCA" on page 478

db2Load - Load

Loads data into a DB2 table. Data residing on the server may be in the form of a file, cursor, tape, or named pipe. Data residing on a remotely connected client may be in the form of a fully qualified file, a cursor, or named pipe. The load utility does not support loading data at the hierarchy level.

Authorization:

One of the following:

- *sysadm*
- *dbadm*
- load authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.

Note: In general, all load processes and all DB2 server processes are owned by the instance owner. All of these processes use the identification of the instance owner to access needed files. Therefore, the instance owner must have read access to the input files, regardless of who invokes the command.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  db2Load (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2PartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
} db2LoadStruct;

typedef SQL_STRUCTURE db2LoadIn
{
   db2Uint64                          iRowcount;
   db2Uint64                          iRestartcount;
   char                               *piUseTablespace;
   db2Uint32                          iSavecount;
   db2Uint32                          iDataBufferSize;
   db2Uint32                          iSortBufferSize;
   db2Uint32                          iWarningcount;
   db2Uint16                          iHoldQuiesce;
   db2Uint16                          iCpuParallelism;
   db2Uint16                          iDiskParallelism;
   db2Uint16                          iNonrecoverable;
   db2Uint16                          iIndexingMode;
   db2Uint16                          iAccessLevel;
   db2Uint16                          iLockWithForce;
   db2Uint16                          iCheckPending;
   char                               iRestartphase;
   char                               iStatsOpt;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64                          oRowsRead;
   db2Uint64                          oRowsSkipped;
   db2Uint64                          oRowsLoaded;
   db2Uint64                          oRowsRejected;
   db2Uint64                          oRowsDeleted;
   db2Uint64                          oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2PartLoadIn
{
   char                               *piHostname;
   char                               *piFileTransferCmd;
   char                               *piPartFileLocation;
   struct db2LoadNodeList             *piOutputNodes;
   struct db2LoadNodeList             *piPartitioningNodes;
   db2Uint16                          *piMode;
   db2Uint16                          *piMaxNumPartAgents;
   db2Uint16                          *piIsolatePartErrs;
   db2Uint16                          *piStatusInterval;
   struct db2LoadPortRange            *piPortRange;
   db2Uint16                          *piCheckTruncation;
   char                               *piMapFileInput;
   char                               *piMapFileOutput;
   db2Uint16                          *piTrace;
   db2Uint16                          *piNewline;
   char                               *piDistfile;
   db2Uint16                          *piOmitHeader;
   SQL_PDB_NODE_TYPE                  *piRunStatDBPartNum;
} db2PartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE                  *piNodeList;
   db2Uint16                          iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16                          iPortMin;
   db2Uint16                          iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64                          oRowsRdPartAgents;
   db2Uint64                          oRowsRejPartAgents;
   db2Uint64                          oRowsPartitioned;
   struct db2LoadAgentInfo            *poAgentInfoList;
   db2Uint32                          iMaxAgentInfoEntries;
   db2Uint32                          oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32                           oSqlcode;
   db2Uint32                          oTableState;
   SQL_PDB_NODE_TYPE                  oNodeNum;
   db2Uint16                          oAgentType;
} db2LoadAgentInfo;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
db2gLoad (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2gPartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   db2Uint16 iFileTypeLen;
   db2Uint16 iLocalMsgFileLen;
   db2Uint16 iTempFilesPathLen;
} db2gLoadStruct;

typedef SQL_STRUCTURE db2LoadIn
{
   db2Uint64                          iRowcount;
   db2Uint64                          iRestartcount;
   char                               *piUseTablespace;
   db2Uint32                          iSavecount;
   db2Uint32                          iDataBufferSize;
   db2Uint32                           iSortBufferSize;
   db2Uint32                           iWarningcount;
   db2Uint16                           iHoldQuiesce;
   db2Uint16                           iCpuParallelism;
   db2Uint16                           iDiskParallelism;
   db2Uint16                           iNonrecoverable;
   db2Uint16                           iIndexingMode;
   db2Uint16                           iAccessLevel;
   db2Uint16                           iLockWithForce;
   db2Uint16                           iCheckPending;
   char                                iRestartphase;
   char                                iStatsOpt;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64                           oRowsRead;
   db2Uint64                           oRowsSkipped;
   db2Uint64                           oRowsLoaded;
   db2Uint64                           oRowsRejected;
   db2Uint64                           oRowsDeleted;
   db2Uint64                           oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2gPartLoadIn
{
   char                                *piHostname;
   char                                *piFileTransferCmd;
   char                                *piPartFileLocation;
   struct db2LoadNodeList              *piOutputNodes;
   struct db2LoadNodeList              *piPartitioningNodes;
   db2Uint16                           *piMode;
   db2Uint16                           *piMaxNumPartAgents;
   db2Uint16                           *piIsolatePartErrs;
   db2Uint16                           *piStatusInterval;
   struct db2LoadPortRange             *piPortRange;
   db2Uint16                           *piCheckTruncation;
   char                                *piMapFileInput;
   char                                *piMapFileOutput;
   db2Uint16                           *piTrace;
   db2Uint16                           *piNewline;
   char                                *piDistfile;
   db2Uint16                           *piOmitHeader;
   SQL_PDB_NODE_TYPE                   *piRunStatDBPartNum;
   db2Uint16                           iHostnameLen;
   db2Uint16                           iFileTransferLen;
   db2Uint16                           iPartFileLocLen;
   db2Uint16                           iMapFileInputLen;
   db2Uint16                           iMapFileOutputLen;
   db2Uint16                           iDistfileLen;
} db2gPartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE                   *piNodeList;
   db2Uint16                           iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16                           iPortMin;
   db2Uint16                           iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64                           oRowsRdPartAgents;
   db2Uint64                           oRowsRejPartAgents;
   db2Uint64                           oRowsPartitioned;
   struct db2LoadAgentInfo             *poAgentInfoList;
   db2Uint32                           iMaxAgentInfoEntries;
   db2Uint32                           oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32                            oSqlcode;
   db2Uint32                           oTableState;
   SQL_PDB_NODE_TYPE                   oNodeNum;
   db2Uint16                           oAgentType;
} db2LoadAgentInfo;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2LoadStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piSourceList
Input. A pointer to an *sqlu_media_list* structure used to provide a list of source files, devices, vendors, pipes, or SQL statements.

The information provided in this structure depends on the value of the *media_type* field. Valid values (defined in `sqlutil`) are:

SQLU_SQL_STMT
If the *media_type* field is set to this value, the caller provides an SQL query through the *pStatement* field of the target field. The *pStatement* field is of type *sqlu_statement_entry*. The sessions field must be set to the value of 1, since the load utility only accepts a single SQL query per load.

SQLU_SERVER_LOCATION
If the *media_type* field is set to this value, the caller provides information through *sqlu_location_entry* structures. The *sessions* field indicates the number of *sqlu_location_entry* structures provided. This is used for files, devices, and named pipes.

SQLU_CLIENT_LOCATION
If the *media_type* field is set to this value, the caller provides information through *sqlu_location_entry* structures. The sessions field indicates the number of *sqlu_location_entry* structures provided. This is used for fully qualified files and named pipes. Note that this *media_type* is only valid if the API is being called via a remotely connected client.

SQLU_TSM_MEDIA
If the *media_type* field is set to this value, the *sqlu_vendor* structure is used, where *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

SQLU_OTHER_MEDIA
If the *media_type* field is set to this value, the *sqlu_vendor* structure is used, where *shr_lib* is the shared library name, and *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

piLobPathList
Input. A pointer to an *sqlu_media_list* structure. For IXF, ASC, and DEL file types, a list of fully qualified paths or devices to identify the location of the individual LOB files to be loaded. The file names are found in the IXF, ASC, or DEL files, and are appended to the paths provided.

The information provided in this structure depends on the value of the *media_type* field. Valid values (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
If set to this value, the caller provides information through *sqlu_media_entry* structures. The *sessions* field indicates the number of *sqlu_media_entry* structures provided.

SQLU_TSM_MEDIA
If set to this value, the *sqlu_vendor* structure is used, where *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

SQLU_OTHER_MEDIA
If set to this value, the *sqlu_vendor* structure is used, where *shr_lib* is the shared library name, and *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

piDataDescriptor
Input. Pointer to an *sqldcol* structure containing information about the columns being selected for loading from the external file.

If the *pFileType* parameter is set to `SQL_ASC`, the *dcolmeth* field of this structure must either be set to `SQL_METH_L` or be set to `SQL_METH_D` and specifies a file name with POSITIONSFILE *pFileTypeMod* modifier which contains starting and ending pairs and null indicator positions. The user specifies the start and end locations for each column to be loaded.

If the file type is `SQL_DEL`, *dcolmeth* can be either `SQL_METH_P` or `SQL_METH_D`. If it is `SQL_METH_P`, the user must provide the source column position. If it is `SQL_METH_D`, the first column in the file is loaded into the first column of the table, and so on.

If the file type is `SQL_IXF`, *dcolmeth* can be one of `SQL_METH_P`, `SQL_METH_D`, or `SQL_METH_N`. The rules for DEL files apply here, except that `SQL_METH_N` indicates that file column names are to be provided in the *sqldcol* structure.

piActionString
Input. Pointer to an *sqlchar* structure, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[FOR EXCEPTION e_tbname]"
```

INSERT
Adds the loaded data to the table without changing the existing table data.

REPLACE
Deletes all existing data from the table, and inserts the loaded data. The table definition and the index definitions are not changed.

RESTART
Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

TERMINATE
Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects may be marked as invalid, in which case index rebuild will automatically take place at next access). If the table spaces in which the table resides are not in load pending state, this option does not affect the state of the table spaces.

The load terminate option will not remove a backup pending state from table spaces.

tbname The name of the table into which the data is to be loaded. The table cannot be a system table or a declared temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

(*column_list***)**
A list of table column names into which the data is to be inserted. The column names must be separated by commas. If a name contains spaces or lowercase characters, it must be enclosed by quotation marks.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the LOAD command.

FOR EXCEPTION *e_tbname*
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a

primary key index is copied. DATALINK exceptions are also captured in the exception table.

piFileType
Input. A string that indicates the format of the input data source. Supported external formats (defined in `sqlutil`) are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integrated Exchange Format, the preferred method for exporting data from a table so that it can be loaded later into the same table or into another database manager table.

SQL_CURSOR
An SQL query. The *sqlu_media_list* structure passed in through the *piSourceList* parameter is of type SQLU_SQL_STMT, and refers to an actual SQL query and not a cursor declared against one.

piFileTypeMod
Input. A pointer to the *sqlchar* structure, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

piLocalMsgFileName
Input. A string containing the name of a local file to which output messages are to be written.

piTempFilesPath
Input. A string containing the path name to be used on the server for temporary files. Temporary files are created to store messages, consistency points, and delete phase information.

piVendorSortWorkPaths
Input. A pointer to the *sqlu_media_list* structure which specifies the Vendor Sort work directories.

piCopyTargetList
Input. A pointer to an *sqlu_media_list* structure used (if a copy image

is to be created) to provide a list of target paths, devices, or a shared library to which the copy image is to be written.

The values provided in this structure depend on the value of the *media_type* field. Valid values for this field (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
If the copy is to be written to local media, set the *media_type* to this value and provide information about the targets in *sqlu_media_entry* structures. The *sessions* field specifies the number of *sqlu_media_entry* structures provided.

SQLU_TSM_MEDIA
If the copy is to be written to TSM, use this value. No further information is required.

SQLU_OTHER_MEDIA
If a vendor product is to be used, use this value and provide further information via an *sqlu_vendor* structure. Set the *shr_lib* field of this structure to the shared library name of the vendor product. Provide only one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field specifies the number of *sqlu_media_entry* structures provided. The load utility will start the sessions with different sequence numbers, but with the same data provided in the one *sqlu_vendor* entry.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. There is a one-to-one ordered correspondence between the elements of this array and the columns being loaded from the data file. That is, the number of elements must equal the *dcolnum* field of the *pDataDescriptor* parameter. Each element of the array contains a number identifying a location in the data file that is to be used as a NULL indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified location in the data file must contain a `Y` or an `N`. A `Y` indicates that the table column data is NULL, and `N` indicates that the table column data is not NULL.

piLoadInfoIn
Input. A pointer to the *db2LoadIn* structure.

poLoadInfoOut
Input. A pointer to the *db2LoadOut* structure.

piPartLoadInfoIn
Input. A pointer to the *db2PartLoadIn* structure.

poPartLoadInfoOut
Output. A pointer to the *db2PartLoadOut* structure.

iCallerAction
Input. An action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. This value (or SQLU_NOINTERRUPT) must be used on the first call to the API.

SQLU_NOINTERRUPT
Initial call. Do not suspend processing. This value (or SQLU_INITIAL) must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested load operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_ABORT
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_RESTART
Restart processing.

SQLU_DEVICE_TERMINATE
Terminate a single device. This option should be specified if the utility is to stop reading data from the device, but further processing of the data is to be done.

iFileTypeLen
Input. Specifies the length in bytes of *iFileType*.

iLocalMsgFileLen
Input. Specifies the length in bytes of *iLocalMsgFileName*.

iTempFilesPathLen
Input. Specifies the length in bytes of *iTempFilesPath*.

iRowcount
Input. The number of physical records to be loaded. Allows a user to load only the first *rowcnt* rows in a file.

iRestartcount
Input. Reserved for future use.

piUseTablespace
Input. If the indexes are being rebuilt, a shadow copy of the index is built in tablespace *iUseTablespaceName* and copied over to the original tablespace at the end of the load. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same tablespace as the index object.

If the shadow copy is created in the same tablespace as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different tablespace from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load.

This field is ignored if *iAccessLevel* is SQLU_ALLOW_NO_ACCESS.

This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

iSavecount
The number of records to load before establishing a consistency point. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using *db2LoadQuery - Load Query*. If the value of *savecnt* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is `0`, meaning that no consistency points will be established, unless necessary.

iDataBufferSize
The number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the required minimum is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

iSortBufferSize
Input. This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the *iIndexingMode* parameter is not specified as SQLU_INX_DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory used by LOAD without changing the value of SORTHEAP, which would also affect general query processing.

iWarningcount
Input. Stops the load operation after *warningcnt* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If *warningcnt* is `0`, or this option is not specified, the load operation will continue regardless of the number of warnings issued.

If the load operation is stopped because the threshold of warnings was exceeded, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

iHoldQuiesce
Input. A flag whose value is set to `TRUE` if the utility is to leave the table in quiesced exclusive state after the load, and to `FALSE` if it is not.

iCpuParallelism
Input. The number of processes or threads that the load utility will spawn for parsing, converting and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, the load utility uses an intelligent default value at run time. Note: If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs, or the value specified by the user.

iDiskParallelism
Input. The number of processes or threads that the load utility will spawn for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

iNonrecoverable
Input. Set to SQLU_NON_RECOVERABLE_LOAD if the load transaction is to be marked as non-recoverable, and it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped. With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. Set to SQLU_RECOVERABLE_LOAD if the load transaction is to be marked as recoverable.

iIndexingMode
Input. Specifies the indexing mode. Valid values (defined in `sqlutil`) are:

SQLU_INX_AUTOSELECT
LOAD chooses between REBUILD and INCREMENTAL indexing modes.

SQLU_INX_REBUILD
Rebuild table indexes.

SQLU_INX_INCREMENTAL
Extend existing indexes.

SQLU_INX_DEFERRED
Do not update table indexes.

iAccessLevel
Input. Specifies the access level. Valid values are:

SQLU_ALLOW_NO_ACCESS
Specifies that the load locks the table exclusively.

SQLU_ALLOW_READ_ACCESS
Specifies that the original data in the table (the non-delta portion) should still be visible to readers while the load is in progress. This option is only valid for load appends, such as a load insert, and will be ignored for load replace.

iLockWithForce
Input. A boolean flag. If set to TRUE load will force other applications

as necessary to ensure that it obtains table locks immediately. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

SQLU_ALLOW_NO_ACCESS loads may force conflicting applications at the start of the load operation. At the start of the load the utility may force applications that are attempting to either query or modify the table.

SQLU_ALLOW_READ_ACCESS loads may force conflicting applications at the start or end of the load operation. At the start of the load the load utility may force applications that are attempting to modify the table. At the end of the load the load utility may force applications that are attempting to either query or modify the table.

iCheckPending
Input. Specifies to put the table into check pending state. If SQLU_CP_IMMEDIATE is specified, check pending state will be immediately cascaded to all dependent and descendent tables. If SQLU_CP_DEFERRED is specified, the cascade of check pending state to dependent tables will be deferred until the target table is checked for integrity violations. SQLU_CP_DEFERRED is the default if the option is not specified.

iRestartphase
Input. The restart phase.

iStatsOpt
Input. Granularity of statistics to collect. Valid values are:

SQLU_STATS_NONE
No statistics to be gathered.

SQL_STATS_INDEX
Basic index statistics without table statistics.

SQL_STATS_TABLE
Basic table statistics without index statistics.

SQL_STATS_BOTH
Basic table and index statistics.

SQL_STATS_EXTTABLE_ONLY
Table and distribution statistics.

SQL_STATS_EXTTABLE_INDEX
Table and distribution statistics and basic index statistics.

SQL_STATS_EXTINDEX_ONLY
Extended statistics for indexes only.

SQL_STATS_EXTINDEX_TABLE
Extended statistics for indexes and basic table statistics.

SQL_STATS_ALL
Extended index statistics with table and distribution statistics.

oRowsRead
Output. Number of records read during the load operation.

oRowsSkipped
Output. Number of records skipped before the load operation begins.

oRowsLoaded
Output. Number of rows loaded into the target table.

oRowsRejected
Output. Number of records that could not be loaded.

oRowsDeleted
Output. Number of duplicate rows deleted.

oRowsCommitted
Output. The total number of processed records: the number of records loaded successfully and committed to the database, plus the number of skipped and rejected records.

piHostname
Input. The hostname for the *iFileTransferCmd* parameter. If NULL, the hostname will default to "nohost".

piFileTransferCmd
Input. File transfer command parameter. If not required, it must be set to NULL. See the Data Movement Guide for a full description of this parameter.

piPartFileLocation
Input. In PARTITION_ONLY, LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the fully qualified location of the partitioned files. This location must exist on each partition specified by the *piOutputNodes* option.

For file types other than SQL_CURSOR, the following rules apply: In PARTITION_ONLY mode, if the value of this parameter is NULL, the current directory will be used as the location where load will create the partitioned files. For LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, if this parameter is NULL, the location of the input files will be taken as the path prefix of the input file name specified in the piSourceList parameter, as long as that name is fully-qualified; otherwise, the current directory will be used as the location of the input files.

For the SQL_CURSOR file type, this parameter cannot by NULL and the location does not refer to a path, but to a fully qualified file name. This will be the fully qualified base file name of the partitioned files that are created on each output partition for PARTITION_ONLY mode, or the location of the files to be read from each partition for LOAD_ONLY mode.

For PARTITION_ONLY mode, multiple files may be created with the specified base name if there are LOB columns in the target table.

piOutputNodes
Input. The list of Load output partitions. A NULL indicates that all nodes on which the target table is defined.

piPartitioningNodes
Input. The list of partitioning nodes. A NULL indicates the default. Refer to the Load command in the Data Movement Guide and Reference for a description of how the default is determined.

piMode
Input. Specifies the load mode for partitioned databases. Valid values (defined in db2ApiDf) are:

PARTITION_AND_LOAD
Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

PARTITION_ONLY
Data is partitioned (perhaps in parallel) and the output is written to files in a specified location on each loading partition. See the description of the *piPartFileLocation* parameter for a description of how to specify the location of the output files.

LOAD_ONLY
Data is assumed to be already partitioned; the partition process is skipped, and the data is loaded simultaneously on the corresponding database partitions. See the description of the *piPartFileLocation* parameter for a description of how to specify the location of the input files.

LOAD_ONLY_VERIFY_PART
Data is assumed to be already partitioned, but the data file does not contain a partition header. The partitioning process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct partition. Rows containing partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If partition violations exist on a particular

loading partition, a single warning will be written to the load message file for that partition. See the description of the *piPartFileLocation* parameter for a description of how to specify the location of the input files.

ANALYZE
An optimal partitioning map with even distribution across all database partitions is generated.

piMaxNumPartAgents
Input. The maximum number of partitioning agents. A NULL value indicates the default, which is 25.

piIsolatePartErrs
Input. Indicates how the load operation will react to errors that occur on individual partitions. Valid values (defined in `db2ApiDf`) are:

DB2LOAD_SETUP_ERRS_ONLY
In this mode, errors that occur on a partition during setup, such as problems accessing a partition or problems accessing a table space or table on a partition, will cause the load operation to stop on the failing partitions but to continue on the remaining partitions. Errors that occur on a partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each partition.

DB2LOAD_LOAD_ERRS_ONLY
In this mode, errors that occur on a partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining partitions until a failure occurs or until all the data is loaded. On the partitions where all of the data was loaded, the data will not be visible following the load operation. Because of the errors in the other partitions the transaction will be aborted. Data on all of the partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the partitions where the load operation completed and resume the load operation on partitions that experienced an error.

Note: This mode cannot be used when *iAccessLevel* is set to SQLU_ALLOW_READ_ACCESS or a copy target is specified.

DB2LOAD_SETUP_AND_LOAD_ERRS
In this mode, partition-level errors during setup or loading data cause processing to stop only on the affected partitions.

As with the DB2LOAD_LOAD_ERRS_ONLY mode, when partition errors do occur while data is being loaded, the data on all partitions will remain invisible until a load restart operation is performed.

Note: This mode cannot be used when *iAccessLevel* is set to SQLU_ALLOW_READ_ACCESS or a copy target is specified.

DB2LOAD_NO_ISOLATION
Any error during the Load operation causes the transaction to be aborted.

If this parameter is NULL, it will default to DB2LOAD_LOAD_ERRS_ONLY, unless *iAccessLevel* is set to SQLU_ALLOW_READ_ACCESS or a copy target is specified, in which case the default is DB2LOAD_NO_ISOLATION.

piStatusInterval
Input. Specifies the number of megabytes (MB) of data to load before generating a progress message. Valid values are whole numbers in the range of 1 to 4000. If NULL is specified, a default value of 100 will be used.

piPortRange
Input. The TCP port range for internal communication. If NULL, the port range used will be 6000-6063.

piCheckTruncation
Input. Causes Load to check for record truncation at Input/Output. Valid values are TRUE and FALSE. If NULL, the default is FALSE.

piMapFileInput
Input. Partition map input filename. If the mode is not ANALYZE, this parameter should be set to NULL. If the mode is ANALYZE, this parameter must be specified.

piMapFileOutput
Input. Partition map output filename. The rules for piMapFileInput apply here as well.

piTrace
Input. Specifies the number of records to trace when you need to review a dump of all the data conversion process and the output of hashing values. If NULL, the number of records defaults to 0.

piNewline
Input. Forces Load to check for newline characters at end of ASC data records if RECLEN file type modifier is also specified. Possible values are TRUE and FALSE. If NULL, the value defaults to FALSE.

piDistfile
Input. Name of the partition distribution file. If a NULL is specified, the value defaults to ″DISTFILE″.

piOmitHeader
Input. Indicates that a partition map header should not be included in the partition file when using DB2LOAD_PARTITION_ONLY mode. Possible values are TRUE and FALSE. If NULL, the default is FALSE.

piRunStatDBPartNum
Specifies the database partition on which to collect statistics. The default value is the first database partition in the output partition list.

iHostnameLen
Input. The length in bytes of *piHostname*.

iFileTransferLen
Input. The length in bytes of *piFileTransferCmd*.

iPartFileLocLen
Input. The length in bytes of *piPartFileLocation*.

iMapFileInputLen
Input. The length in bytes of *piMapFileInput*.

iMapFileOutputLen
Input. The length in bytes of *piMapFileOutput*.

iDistfileLen
Input. The length in bytes of *piDistfile*.

piNodeList
Input. An array of node numbers.

iNumNodes
Input. The number of nodes in the *piNodeList* array. A 0 indicates the default, which is all nodes on which the target table is defined.

iPortMin
Input. Lower port number.

iPortMax
Input. Higher port number.

oRowsRdPartAgents
Output. Total number of rows read by all partitioning agents.

oRowsRejPartAgents
Output. Total number of rows rejected by all partitioning agents.

oRowsPartitioned
Output. Total number of rows partitioned by all partitioning agents.

poAgentInfoList
Output. During a load operation into a partitioned database, the following load processing entities may be involved: load agents, partitioning agents, pre-partitioing agents, file transfer command agents and load-to-file agents (these are described in the Data Movement Guide). The purpose of the *poAgentInfoList* output parameter is to return to the caller information about each load agent that participated in a load operation. Each entry in the list contains the following information:

- oAgentType. A tag indicating what kind of load agent the entry describes.
- oNodeNum. The number of the partition on which the agent executed.
- oSqlcode. The final sqlcode resulting from the agent's processing.
- oTableState. The final status of the table on the partition on which the agent executed (relevant only for load agents).

It is up to the caller of the API to allocate memory for this list prior to calling the API. The caller should also indicate the number of entries for which they allocated memory in the *iMaxAgentInfoEntries* parameter. If the caller sets *poAgentInfoList* to NULL or sets *iMaxAgentInfoEntries* to 0, then no information will be returned about the load agents.

iMaxAgentInfoEntries
Input. The maximum number of agent information entries allocated by the user for *poAgentInfoList*. In general, setting this parameter to 3 times the number of partitions involved in the load operation should be sufficient.

oNumAgentInfoEntries
Output. The actual number of agent information entries produced by the load operation. This number of entries will be returned to the user in the *poAgentInfoList* parameter as long as *iMaxAgentInfoEntries* is greater than or equal to *oNumAgentInfoEntries*. If *iMaxAgentInfoEntries* is less than *oNumAgentInfoEntries*, then the number of entries returned in *poAgentInfoList* is equal to *iMaxAgentInfoEntries*.

oSqlcode
Output. The final sqlcode resulting from the agent's processing.

oTableState
Output. The purpose of this output parameter is not to report every possible state of the table after the load operation. Rather, its purpose is to report only a small subset of possible tablestates in order to give

the caller a general idea of what happened to the table during load processing. This value is relevant for load agents only. The possible values are:

DB2LOADQUERY_NORMAL
Indicates that the load completed successfully on the partition and the table was taken out of the LOAD IN PROGRESS (or LOAD PENDING) state. In this case, the table still could be in CHECK PENDING state due to the need for further constraints processing, but this will not reported as this is normal.

DB2LOADQUERY_UNCHANGED
Indicates that the load job aborted processing due to an error but did not yet change the state of the table on the partition from whatever state it was in prior to calling db2Load. It is not necessary to perform a load restart or terminate operation on such partitions.

DB2LOADQUERY_LOADPENDING
Indicates that the load job aborted during processing but left the table on the partition in the LOAD PENDING state, indicating that the load job on that partition must be either terminated or restarted.

oNodeNum
Output. The number of the partition on which the agent executed.

oAgentType
Output. The agent type. Valid values (defined in `db2ApiDf`) are :

DB2LOAD_LOAD_AGENT

DB2LOAD_PARTITIONING_AGENT

DB2LOAD_PRE_PARTITIONING_AGENT

DB2LOAD_FILE_TRANSFER_AGENT

DB2LOAD_LOAD_TO_FILE_AGENT

Usage notes:

Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted.

The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update summary tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in check pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in check pending state. Issue the SET INTEGRITY statement to take the tables out of check pending state. Load operations cannot be carried out on replicated summary tables.

For clustering indexes, the data should be sorted on the clustering index prior to loading. The data need not be sorted when loading into an multi-dimensionally clustered (MDC) table.

DB2 Data Links Manager Considerations

For each DATALINK column, there can be one column specification within parentheses. Each column specification consists of one or more of DL_LINKTYPE, *prefix* and a DL_URL_SUFFIX specification. The *prefix* information can be either DL_URL_REPLACE_PREFIX, or the DL_URL_DEFAULT_PREFIX specification.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns as found within the insert-column list (if specified by INSERT INTO (insert-column, ...)), or within the table definition (if insert-column is not specified).

For example, if a table has columns C1, C2, C3, C4, and C5, and among them only columns C2 and C5 are of type DATALINK, and the insert-column list is (C1, C5, C3, C2), there should be two DATALINK column specifications. The first column specification will be for C5, and the second column specification will be for C2. If an insert-column list is not specified, the first column specification will be for C2, and the second column specification will be for C5.

If there are multiple DATALINK columns, and some columns do not need any particular specification, the column specification should have at least the parentheses to unambiguously identify the order of specifications. If there are no specifications for any of the columns, the entire list of empty parentheses can be dropped. Thus, in cases where the defaults are satisfactory, there need not be any DATALINK specification.

If data is being loaded into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary).

1. Ensure that the DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values. For Distributed File Systems (DFS), ensure that the DB2 Data Links Managers within the target cell are registered.

2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers, all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.
5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file. For DFS, register the cells at the target configuration referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

The connection between DB2 and the Data Links server may fail while running the load utility, causing the load operation to fail. If this occurs:

1. Start the Data Links server and the DB2 Data Links Manager.
2. Invoke a load restart operation.

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns.

Representation of DATALINK Information in an Input File

The LINKTYPE (currently only URL is supported) is not specified as part of DATALINK information. The LINKTYPE is specified in the LOAD or the IMPORT command, and for input files of type PC/IXF, in the appropriate column descriptor records.

The syntax of DATALINK information for a URL LINKTYPE is as follows:

```
>>──┬─────────┬───┬────────────────────────┬──────────────────────────────────────────><
    └─urlname─┘   └─dl_delimiter──comment─┘
```

Note that both *urlname* and *comment* are optional. If neither is provided, the NULL value is assigned.

urlname

The URL name must conform to valid URL syntax.

Notes:

1. Currently "http", "file", "unc", and "dfs" are permitted as a schema name.
2. The prefix (schema, host, and port) of the URL name is optional. For DFS, the prefix refers to the schema cellname filespace-junction portion. If a prefix is not present, it is taken from the DL_URL_DEFAULT_PREFIX or the DL_URL_REPLACE_PREFIX

specification of the load or the import utility. If none of these is specified, the prefix defaults to ″file://localhost″. Thus, in the case of local files, the file name with full path name can be entered as the URL name, without the need for a DATALINK column specification within the LOAD or the IMPORT command.

3. Prefixes, even if present in URL names, are overridden by a different prefix name on the DL_URL_REPLACE_PREFIX specification during a load or import operation.
4. The ″path″ (after appending DL_URL_SUFFIX, if specified) is the full path name of the remote file in the remote server. Relative path names are not allowed. The http server default path-prefix is not taken into account.

dl_delimiter

For the delimited ASCII (DEL) file format, a character specified via the `dldel` modifier, or defaulted to on the LOAD or the IMPORT command. For the non-delimited ASCII (ASC) file format, this should correspond to the character sequence `\;` (a backslash followed by a semicolon). Whitespace characters (blanks, tabs, and so on) are permitted before and after the value specified for this parameter.

comment

The comment portion of a DATALINK value. If specified for the delimited ASCII (DEL) file format, the *comment* text must be enclosed by the character string delimiter, which is double quotation marks (″) by default. This character string delimiter can be overridden by the MODIFIED BY *filetype-mod* specification of the LOAD or the IMPORT command.

If no comment is specified, the comment defaults to a string of length zero.

Following are DATALINK data examples for the delimited ASCII (DEL) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg; "Intro Movie"`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang; "InderPal's Home Page"`

  This is stored with the following parts:
  - scheme = file
  - server = narang

- path = /u/narang
- comment = ″InderPal′s Home Page″

Following are DATALINK data examples for the non-delimited ASCII (ASC) file format:

- `http://www.almaden.ibm.com:80/mrep/intro.mpeg\;Intro Movie`

  This is stored with the following parts:
  - scheme = http
  - server = www.almaden.ibm.com
  - path = /mrep/intro.mpeg
  - comment = ″Intro Movie″
- `file://narang/u/narang\; InderPal's Home Page`

  This is stored with the following parts:
  - scheme = file
  - server = narang
  - path = /u/narang
  - comment = ″InderPal′s Home Page″

Following are DATALINK data examples in which the load or import specification for the column is assumed to be DL_URL_REPLACE_PREFIX (″http://qso″):

- `http://www.almaden.ibm.com/mrep/intro.mpeg`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /mrep/intro.mpeg
  - comment = NULL string
- `/u/me/myfile.ps`

  This is stored with the following parts:
  - schema = http
  - server = qso
  - path = /u/me/myfile.ps
  - comment = NULL string

Table 6. Valid File Type Modifiers (LOAD)

Modifier	Description
All File Formats	

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
anyorder	This modifier is used in conjunction with the *cpu_parallelism* parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of *cpu_parallelism* is 1, this option is ignored. This option is not supported if SAVECOUNT > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
fastparse	Reduced syntax checking is done on user-supplied column values, and performance is enhanced. Tables loaded under this option are guaranteed to be architecturally correct, and the utility is guaranteed to perform sufficient data checking to prevent a segmentation violation or trap. Data that is in correct form will be loaded correctly. For example, if a value of `123qwr4` were to be encountered as a field entry for an integer column in an ASC file, the load utility would ordinarily flag a syntax error, since the value does not represent a valid number. With `fastparse`, a syntax error is not detected, and an arbitrary number is loaded into the integer field. Care must be taken to use this modifier with clean data only. Performance improvements using this option with ASCII data can be quite substantial, but `fastparse` does not significantly enhance performance with PC/IXF data, since IXF is a binary format, and `fastparse` affects parsing and conversion from ASCII to internal forms. This option is not supported in conjunction with the CURSOR filetype.
generatedignore	This modifier informs the load utility that data for all generated columns is present in the data file but should be ignored. This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedmissing` or the `generatedoverride` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated column (not even NULLs). This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedignore` or the `generatedoverride` modifier.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

generatedoverride	This modifier instructs the load utility to accept explicit, non-NULL user-supplied data for all generated columns in the table (contrary to the normal rules for these types of columns). This is useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W).
	Note: When this modifier is used, the table will be placed in CHECK PENDING state. To take the table out of CHECK PENDING state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR <table-name> GENERATED  COLUMN IMMEDIATED UNCHE`
	
	To take the table out of CHECK PENDING state and force verification of the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR <table-name> IMMEDIATE CHECKED`

identityignore	This modifier informs the load utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the `identitymissing` or the `identityoverride` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with either the `identityignore` or the `identityoverride` modifier.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
identityoverride	This modifier should be used only when an identity column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of identity columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the identity column will be rejected (SQL3116W). This modifier cannot be used with either the `identitymissing` or the `identityignore` modifier. **Note:** The load utility will not attempt to maintain or verify the uniqueness of values in the table's identity column when this option is used.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when loading the index. The first entry in a page is added without restriction; subsequent entries are added if the percent free space threshold can be maintained. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement; the registry variable DB2 INDEX FREE takes precedence over indexfreespace. The indexfreespace option affects index leaf pages only.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB values. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column. This option is not supported in conjunction with the CURSOR filetype. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
noheader	Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group). The AutoLoader utility writes a header to each file contributing data to a table in a multiple-partition database partition group. The header includes the database partition number, the partitioning map, and the partitioning key specification. The load utility requires this information to verify that the data is being loaded at the correct database partition. When loading files into a table that exists on a single-partition database partition group, the headers do not exist, and this option causes the load utility to skip the header verification code.
norowwarnings	Suppresses all warnings about rejected rows.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. **Note:** The PCTFREE value of a table determines the amount of free space designated per page. If a pagefreespace value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by pagefreespace overrides the PCTFREE value specified for the table.
totalfreespace=*x*	*x* is an integer greater than or equal to 0 . The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages after the data has been loaded, 20 additional empty pages will be appended. The total number of data pages for the table will be 120. The data pages total does not factor in the number of index pages in the table. This option does not affect the index object. **Note:** If two loads are done with this option specified, the second load will not not reuse the extra space appended to the end by the first load.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: ",," is specified for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
codepage=*x*	<*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters may not coincide with the shift-in and shift-out DBCS characters. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different. This option is not supported in conjunction with the CURSOR filetype.

dateformat="*x*"	*x* is the format of the date in the source file.[a] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M    - Month (one or two digits ranging from 1 - 12)`
	`MM   - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D    - Day (one or two digits ranging from 1 - 31)`
	`DD   - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD  - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

dumpfile = *x*	*x* is the fully qualified (according to the server node) name of an exception file to which rejected rows are written. A maximum of 32KB of data is written per record. Following is an example that shows how to specify a dump file:
	`db2 load from data of del`
	`   modified by dumpfile = /u/user/filename`
	`   insert into table_name`
	Notes:
	1. In a partitioned database environment, the path should be local to the loading node, so that concurrently running load operations do not attempt to write to the same file.
	2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass.
	3. This modifier does not support file names with multiple file extensions. For example,
	`dumpfile = /home/svtdbm6/DUMP.FILE`
	is acceptable to the load utility, but
	`dumpfile = /home/svtdbm6/DUMP.LOAD.FILE`
	is not.

implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

timeformat=*"x"*	*x* is the format of the time in the source file.[a] Valid time elements are:
	```
	H     - Hour (one or two digits ranging from 0 - 12
	           for a 12 hour system, and 0 - 24
	           for a 24 hour system)
	HH    - Hour (two digits ranging from 0 - 12
	           for a 12 hour system, and 0 - 24
	           for a 24 hour system; mutually exclusive
	           with H)
	M     - Minute (one or two digits ranging
	           from 0 - 59)
	MM    - Minute (two digits ranging from 0 - 59;
	           mutually exclusive with M)
	S     - Second (one or two digits ranging
	           from 0 - 59)
	SS    - Second (two digits ranging from 0 - 59;
	           mutually exclusive with S)
	SSSSS - Second of the day after midnight (5 digits
	           ranging from 00000 - 86399; mutually
	           exclusive with other time elements)
	TT    - Meridian indicator (AM or PM)
	```
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	```
	"HH:MM:SS"
	"HH.MM TT"
	"SSSSS"
	```

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[a] Valid time stamp elements are:
	`YYYY   - Year (four digits ranging from 0000 - 9999)`
	`M      - Month (one or two digits ranging from 1 - 12)`
	`MM     - Month (two digits ranging from 1 - 12; mutually exclusive with M, month)`
	`D      - Day (one or two digits ranging from 1 - 31)`
	`DD     - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD    - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H      - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH     - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M      - Minute (one or two digits ranging from 0 - 59)`
	`MM     - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S      - Second (one or two digits ranging from 0 - 59)`
	`SS     - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS  - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999)`
	`TT     - Meridian indicator (AM or PM)`
	
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`insert into schedule`

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.
	ASC (Non-delimited ASCII) File Format
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The `noeofchar` option is assumed. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. **Note:** NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[b] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

packeddecimal	Loads packed-decimal data directly, since the `binarynumerics` modifier does not include the DECIMAL field type.
	
	This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The `noeofchar` option is assumed.
	
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Note: NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on the Windows operating system, the byte order must not be reversed.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with `striptnulls`. These are mutually exclusive options. **Note:** This option replaces the obsolete `t` option, which is supported for back-level compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

zoneddecimal	Loads zoned decimal data, since the BINARYNUMERICS modifier does not include the DECIMAL field type. This option is supported only with positional ASC, using fixed length records specified by the RECLEN option. The NOEOFCHAR option is assumed.
	
	Half-byte sign values can be one of the following:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Supported values for digits are `0x0` to `0x9`.
	
	Supported values for zones are `0x3` and `0xF`.

DEL (Delimited ASCII) File Format	

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[bc]
	
	The single quotation mark (') can also be specified as a character string delimiter as follows:
	`modified by chardel''`
	
	If you wish to explicitly specify the double quotation mark(″) as the character string delimiter, you should specify it as follows:
	`modified by chardel""`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[bc]
datesiso	Date format. Causes all date data values to be loaded in ISO format.
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[bc]

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	**Description**

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 load ... modified by delprioritychar ...`
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [bcd] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.

keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and tailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
	The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file:
	`db2 load from delfile3 of del`
	`   modified by keepblanks`
	`   insert into table1`

nodoubledel	Suppresses recognition of double character delimiters. For more information, see the delimiter restrictions in the import API.
IXF File Format	

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Table 6. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

Notes:

1. The load utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the load operation fails, and an error code is returned.
2. [a] Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Note: Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
3. [b] The character must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
4. [c] The delimeter restrictions in the Import API lists restrictions that apply to the characters that can be used as delimiter overrides.
5. [d] Even if the DATALINK delimiter character is a valid character within the URL syntax, it will lose its special meaning within the scope of the load operation.

Related reference:

- "sqluvqdp - Quiesce Table Spaces for Table" on page 451
- "db2LoadQuery - Load Query" on page 172
- "SQLU-MEDIA-LIST" on page 524
- "sqluexpr - Export" on page 408
- "sqluimpr - Import" on page 424
- "db2DatabaseQuiesce - Database Quiesce" on page 51
- "db2InstanceQuiesce - Instance Quiesce" on page 101

Related samples:

- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbload.sqc -- How to load into a partitioned database (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2LoadQuery - Load Query

Checks the status of a load operation during processing.

Authorization:

None

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2LoadQuery */
/* ... */
SQL_API_RC SQL_API_FN
db2LoadQuery (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  char *piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct *poOutputStruct;
  char *piLocalMessageFile;
} db2LoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
  db2Uint32 oTableState;
} db2LoadQueryOutputStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gLoadQuery */
/* ... */
SQL_API_RC SQL_API_FN
db2gLoadQuery (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  db2Uint32 iStringLen;
  char *piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct *poOutputStruct;
  db2Uint32 iLocalMessageFileLen;
  char *piLocalMessageFile
} db2gLoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
  db2Uint32 oTableState;
} db2LoadQueryOutputStruct;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
: Input. A pointer to the *db2LoadQueryStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iStringType
: Input. Specifies a type for *piString*. Valid values (defined in `db2ApiDf.h`) are:

  DB2LOADQUERY_TABLENAME
  : Specifies a table name for use by the db2LoadQuery API.

iStringLen
: Input. Specifies the length in bytes of *piString*.

piString
: Input. Specifies a temporary files path name or a table name, depending on the value of *iStringType*.

iShowLoadMessages
: Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in `db2ApiDf.h`) are:

  DB2LOADQUERY_SHOW_ALL_MSGS
  : Return all load messages.

  DB2LOADQUERY_SHOW_NO_MSGS
  : Return no load messages.

  DB2LOADQUERY_SHOW_NEW_MSGS
  : Return only messages that have been generated since the last call to this API.

poOutputStruct
: Output. A pointer to the *db2LoadQueryOutputStruct* structure, which contains load summary information. Set to NULL if a summary is not required.

iLocalMessageFileLen
Input. Specifies the length in bytes of *piLocalMessageFile*.

piLocalMessageFile
Input. Specifies the name of a local file to be used for output messages.

oRowsRead
Output. Number of records read so far by the load utility.

oRowsSkipped
Output. Number of records skipped before the load operation began.

oRowsCommitted
Output. Number of rows committed to the target table so far.

oRowsLoaded
Output. Number of rows loaded into the target table so far.

oRowsRejected
Output. Number of rows rejected from the target table so far.

oRowsDeleted
Output. Number of rows deleted from the target table so far (during the delete phase).

oCurrentIndex
Output. Index currently being built (during the build phase).

oCurrentMPPNode
Output. Indicates which database partition server is being queried (for partitioned database environment mode only).

oLoadRestarted
Output. A flag whose value is `TRUE` if the load operation being queried is a load restart operation.

oWhichPhase
Output. Indicates the current phase of the load operation being queried. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_LOAD_PHASE
Load phase.

DB2LOADQUERY_BUILD_PHASE
Build phase.

DB2LOADQUERY_DELETE_PHASE
Delete phase.

oNumTotalIndexes
Output. Total number of indexes to be built (during the build phase).

oWarningCount
Output. Total number of warnings returned so far.

oTableState
Output. The table states. Valid values (as defined in `db2ApiDf`) are:

DB2LOADQUERY_NORMAL
No table states affect the table.

DB2LOADQUERY_CHECK_PENDING
The table has constraints and the constraints have yet to be verified. Use the SET INTEGRITY command to take the table out of the DB2LOADQUERY_CHECK_PENDING state. The load utility puts a table into the DB2LOADQUERY_CHECK_PENDING state when it begins a load on a table with constraints.

DB2LOADQUERY_LOAD_IN_PROGRESS
There is a load actively in progress on this table.

DB2LOADQUERY_LOAD_PENDING
A load has been active on this table but has been aborted before the load could commit. Issue a load terminate, a load restart, or a load replace to bring the table out of the DB2LOADQUERY_LOAD_PENDING state.

DB2LOADQUERY_READ_ACCESS
The table data is available for read access queries. Loads using the DB2LOADQUERY_READ_ACCESS option put the table into Read Access Only state.

DB2LOADQUERY_NOTAVAILABLE
The table is unavailable. The table may only be dropped or it may be restored from a backup. Rollforward through a non-recoverable load will put a table into the unavailable state.

DB2LOADQUERY_NO_LOAD_RESTART
The table is in a partially loaded state that will not allow a load restart. The table will also be in the Load Pending state. Issue a load terminate or a load replace to bring the table out of the No Load Restartable state. The table can be placed in the DB2LOADQUERY_NO_LOAD_RESTART state during a rollforward operation. This can occur if you rollforward to a point in time that is prior to the end of a load operation, or if you roll forward through an aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

Usage notes:

This API reads the status of the load operation on the table specified by *piString*, and writes the status to the file specified by *pLocalMsgFileName*.

Related concepts:

- "Monitoring a Partitioned Database Load Using Load Query" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "SQLCA" on page 478

Related samples:

- "loadqry.sqb -- Query the current status of a load (MF COBOL)"
- "tbload.sqc -- How to load into a partitioned database (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2MonitorSwitches - Get/Update Monitor Switches

Selectively turns on or off switches for groups of monitor data to be collected by the database manager. Returns the current state of these switches for the application issuing the call.

Scope:

This API can return information for the database partition server on the instance, or all database partitions on the instance.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

To display the settings for a remote instance (or a different local instance), it is necessary to first attach to that instance.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2MonitorSwitches */
/* ... */
SQL_API_RC SQL_API_FN
db2MonitorSwitches (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
        struct sqlm_recording_group     *piGroupStates;
        void                            *poBuffer;
        db2Uint32                       iBufferSize;
        db2Uint32                       iReturnData;
        db2Uint32                       iVersion;
        db2int32                        iNodeNumber;
        db2Uint32                       *poOutputFormat;
}db2MonitorSwitchesData;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gMonitorSwitches */
/* ... */
SQL_API_RC SQL_API_FN
db2gMonitorSwitches (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gMonitorSwitchesData
{
  struct sqlm_recording_group *piGroupStates;
  void                        *poBuffer;
  db2Uint32                   iBufferSize;
  db2Uint32                   iReturnData;
  db2Uint32                   iVersion;
  db2int32                    iNodeNumber;
  db2Uint32                   *poOutputFormat;
} db2gMonitorSwitchesData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the *db2MonitorSwitchesStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piGroupStates
Input. A pointer to the *sqlm-recording-group* structure (defined in `sqlmon.h`) containing a list of switches.

poBuffer
A pointer to a buffer where the switch state data will be written.

iBufferSize
Input. Specifies the size of the output buffer.

iReturnData
Input. A flag specifying whether or not the current switch states should be written to the data buffer pointed to by *poBuffer*.

iVersion
Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- `SQLM_DBMON_VERSION1`
- `SQLM_DBMON_VERSION2`
- `SQLM_DBMON_VERSION5`
- `SQLM_DBMON_VERSION5_2`
- `SQLM_DBMON_VERSION6`
- `SQLM_DBMON_VERSION7`
- `SQLM_DBMON_VERSION8`

Note: If `SQLM_DBMON_VERSION1` is specified as the version, the APIs cannot be run remotely.

iNodeNumber
Input. The database partition server where the request is to be sent. Based on this value, the request will be processed for the current database partition server, all database partition servers or a user specified database partition server. Valid values are:

- `SQLM_CURRENT_NODE`
- `SQLM_ALL_NODES`
- *`node value`*

Note: For standalone instances `SQLM_CURRENT_NODE` must be used.

poOutputFormat
The format of the stream returned by the server. It will be one of the following:

SQLM_STREAM_STATIC_FORMAT
Indicates that the switch states are returned in static, pre-Version 7 switch structures.

SQLM_STREAM_DYNAMIC_FORMAT
Indicates that the switches are returned in a self-describing format, similar to the format returned for db2GetSnapshot.

Usage notes:

To obtain the status of the switches at the database manager level, call db2GetSnapshot, specifying `SQMA_DB2` for *OBJ_TYPE* (get snapshot for database manager).

The timestamp switch is unavailable if iVersion is less than `SQLM_DBMON_VERSION8`.

Related reference:
- "db2GetSnapshot - Get Snapshot" on page 73
- "db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot Output Buffer" on page 76
- "db2ResetMonitor - Reset Monitor" on page 205
- "SQLCA" on page 478
- "SQLM-RECORDING-GROUP" on page 517

Related samples:
- "utilsnap.c -- Utilities for the snapshot monitor samples (C)"
- "utilsnap.C -- Utilities for the snapshot monitor samples (C++)"

db2Prune - Prune History File

Deletes entries from the history file or log files from the active log path.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

- *dbadm*

Required connection:

Database. To delete entries from the history file for any database other than the default database, a connection to the database must be established before calling this API.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Prune */
/* ... */
SQL_API_RC SQL_API_FN
db2Prune (
  db2Uint32 version,
  void *pDB2PruneStruct,
  struct sqlca *pSqlca);

typedef struct
{
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2PruneStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2GenPrune */
/* ... */
SQL_API_RC SQL_API_FN
db2GenPrune (
  db2Uint32 version,
  void *pDB2GenPruneStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringLen;
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2GenPruneStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed in as the second parameter, *pDB2PruneStruct*.

pDB2PruneStruct
Input. A pointer to the *db2PruneStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iStringLen
Input. Specifies the length in bytes of *piString*.

piString
Input. A pointer to a string specifying a time stamp or a log sequence number (LSN). The time stamp or part of a time stamp (minimum *yyyy*, or year) is used to select records for deletion. All entries equal to or less than the time stamp will be deleted. A valid time stamp must be provided; there is no default behavior for a NULL parameter.

This parameter can also be used to pass an LSN, so that inactive logs can be pruned.

iEID Input. Specifies a unique identifier that can be used to prune a single entry from the history file.

iCallerAction
Input. Specifies the type of action to be taken. Valid values (defined in `db2ApiDf.h`) are:

DB2PRUNE_ACTION_HISTORY
Remove history file entries.

DB2PRUNE_ACTION_LOG
Remove log files from the active log path.

iOptions
Input. Valid values (defined in `db2ApiDf.h`) are:

DB2PRUNE_OPTION_FORCE
Force the removal of the last backup.

DB2PRUNE_OPTION_LSNSTRING
Specify that the value of *piString* is an LSN, used when a caller action of `DB2PRUNE_ACTION_LOG` is specified.

REXX API syntax:

```
PRUNE RECOVERY HISTORY BEFORE :timestamp [WITH FORCE OPTION]
```

REXX API parameters:

timestamp
A host variable containing a time stamp. All entries with time stamps equal to or less than the time stamp provided are deleted from the history file.

WITH FORCE OPTION
If specified, the history file will be pruned according to the time stamp specified, even if some entries from the most recent restore set are deleted from the file. If not specified, the most recent restore set will be kept, even if the time stamp is less than or equal to the time stamp specified as input.

Usage notes:

Pruning the history file does not delete the actual backup or load files. The user must manually delete these files to free up the space they consume on storage media.

CAUTION:
If the latest full database backup is deleted from the media (in addition to being pruned from the history file), the user must ensure that all table spaces, including the catalog table space and the user table spaces, are backed up. Failure to do so may result in a database that cannot be recovered, or the loss of some portion of the user data in the database.

Related reference:
- "db2HistoryUpdate - Update History File" on page 90
- "db2HistoryOpenScan - Open History File Scan" on page 86
- "db2HistoryGetEntry - Get Next History File Entry" on page 82
- "db2HistoryCloseScan - Close History File Scan" on page 81
- "SQLCA" on page 478

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2QuerySatelliteProgress - Query Satellite Sync

Checks on the status of a satellite synchronization session.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2QuerySatelliteProgress */
/* ... */
SQL_API_RC SQL_API_FN
  db2QuerySatelliteProgress (
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
  db2int32 oStep;
  db2int32 oSubstep;
  db2int32 oNumSubsteps;
  db2int32 oScriptStep;
  db2int32 oNumScriptSteps;
  char *poDescription;
  char *poError;
  char *poProgressLog;
} db2QuerySatelliteProgressStruct;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
: Input. A pointer to the *db2QuerySatelliteProgressStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

oStep Output. The current step of the synchronization session (defined in `db2ApiDf.h`).

oSubstep
Output. If the synchronization step (*oStep*) can be broken down into substeps, this will be the current substep.

oNumSubsteps
Output. If there exists a substep (*oSubstep*) for the current step of the synchronization session, this will be the total number of substeps that comprise the synchronization step.

oScriptStep
Output. If the current substep is the execution of a script, this parameter reports on the progress of the script execution, if available.

oNumScriptSteps
Output. If a script step is reported, this parameter contains the total number of steps that comprise the script's execution.

poDescription
Output. A description of the state of the satellite's synchronization session.

poError
Output. If the synchronization session is in error, a description of the error is passed by this parameter.

poProgressLog
Output. The entire log of the satellite's synchronization session is returned by this parameter.

Related reference:
- "SQLCA" on page 478

db2ReadLog - Asynchronous Read Log

Extract log records from the DB2 UDB database logs and the Log Manager for current log state information. This API can only be used with recoverable databases. A database is recoverable if it is configured with *logretain* set to `RECOVERY` or *userexit* set to `ON`.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLog */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLog (
  db2Uint32 versionNumber,
  void *pDB2ReadLogStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReadLogStruct
{
  db2Uint32                   iCallerAction;
  SQLU_LSN                    *piStartLSN;
  SQLU_LSN                    *piEndLSN;
  char                        *poLogBuffer;
  db2Uint32                   iLogBufferSize;
  db2Uint32                   iFilterOption;
  db2ReadLogInfoStruct        *poReadLogInfo;

typedef SQL_STRUCTURE db2ReadLogInfoStruct
{
  SQLU_LSN                    initialLSN;
  SQLU_LSN                    firstReadLSN;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  SQLU_LSN                    firstReusedLSN;
  db2Uint32                   timeOfLSNReuse;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogInfoStruct;

typedef SQL_STRUCTURE db2TimeOfLog
{
  db2Uint32                   seconds;
  db2Uint32                   accuracy;
} db2TimeOfLog;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter, *pDB2ReadLogStruct*.

pDB2ReadLogStruct
Input. A pointer to the *db2ReadLogStruct*.

pSqlca
Output. A pointer to the *sqlca* structure.

iCallerAction
Input. Specifies the action to be performed.

DB2READLOG_READ
Read the database log from the starting log sequence to the ending log sequence number and return log records within this range.

DB2READLOG_READ_SINGLE
Read a single log record (propagatable or not) identified by the starting log sequence number.

DB2READLOG_QUERY
Query the database log. Results of the query will be sent back via the *db2ReadLogInfoStruct* structure.

piStartLsn
Input. The starting log sequence number specifies the starting relative byte address for the reading of the log. This value must be the start of an actual log record.

piEndLsn
Input. The ending log sequence number specifies the ending relative byte address for the reading of the log. This value must be greater than *startLsn*, and does not need to be the end of an actual log record.

poLogBuffer
Output. The buffer where all the propagatable log records read within the specified range are stored sequentially. This buffer must be large enough to hold a single log record. As a guideline, this buffer should be a minimum of 32 bytes. Its maximum size is dependent on the size of the requested range. Each log record in the buffer is prefixed by a six byte log sequence number (LSN), representing the LSN of the following log record.

iLogBufferSize
Input. Specifies the size, in bytes, of the log buffer.

iFilterOption
Input. Specifies the level of log record filtering to be used when reading the log records. Valid values are:

DB2READLOG_FILTER_OFF
Read all log records in the given LSN range.

DB2READLOG_FILTER_ON
Reads only log records in the given LSN range marked as propagatable. This is the traditional behaviors of the asynchronous log read API.

poReadLogInfo
Output. A structure detailing information regarding the call and the database log.

Usage notes:

If the requested action is to read the log, the caller will provide a log sequence number range and a buffer to hold the log records. This API reads the log sequentially, bounded by the requested LSN range, and returns log records associated with tables having the DATA CAPTURE option CHANGES, and a db2ReadLogInfoStruct structure with the current active log information. If the requested action is query, the API returns an db2ReadLogInfoStruct structure with the current active log information.

To use the Asynchronous Log Reader, first query the database log for a valid starting LSN. Following the query call, the read log information structure (db2ReadLogInfoStruct) will contain a valid starting LSN (in the initialLSN member), to be used on a read call. The value used as the ending LSN on a read can be one of the following:
- A value greater than initialLSN
- `FFFF FFFF FFFF`, which is interpreted by the asynchronous log reader as the end of the current log.

The propagatable log records read within the starting and ending LSN range are returned in the log buffer. A log record does not contain its LSN; it is contained in the buffer before the actual log record. Descriptions of the various DB2 log records returned by db2ReadLog the DB2 UDB Log Records section.

To read the next sequential log record after the initial read, use the nextStartLSN field returned in the db2ReadLogStruct structure. Resubmit the call, with this new starting LSN and a valid ending LSN. The next block of records is then read. An *sqlca* code of SQLU_RLOG_READ_TO_CURRENT means that the log reader has read to the end of the current active log.

Related reference:
- "SQLCA" on page 478
- "db2Reorg - Reorganize" on page 196

Related samples:

- “dbrecov.sqc -- How to recover a database (C)”
- “dbrecov.sqC -- How to recover a database (C++)”

db2ReadLogNoConn - Read Log Without a Database Connection

Extract log records from the DB2 UDB database logs and query the Log Manager for current log state information. Prior to using this API, use db2ReadLogNoConnInit to allocate the memory that is passed as an input parameter to this API. After using this API, use db2ReadLogNoConnTerm to deallocate the memory.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConn */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConn (
  db2Uint32 versionNumber,
  void *pDB2ReadLogNoConnStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnStruct
{
  db2Uint32                  iCallerAction;
  SQLU_LSN                   *piStartLSN;
  SQLU_LSN                   *piEndLSN;
  char                       *poLogBuffer;
  db2Uint32                  iLogBufferSize;
  char                       *piReadLogMemPtr;
  db2ReadLogNoConnInfoStruct  *poReadLogInfo;
} db2ReadLogNoConnStruct;

typedef SQL_STRUCTURE db2ReadLogNoConnInfoStruct
{
  SQLU_LSN                    firstAvailableLSN;
  SQLU_LSN                    firstReadLSN;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  db2Uint32                   lastLogFullyRead;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogNoConnInfoStruct;

/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed as the second parameter *pDB2ReadLogNoConnStruct*.

pDB2ReadLogNoConnStruct
Input. A pointer to the *db2ReadLogNoConnStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iCallerAction
Input. Specifies the action to be performed. Valid values are:

DB2READLOG_READ
Read the database log from the starting log sequence to the ending log sequence number and return log records within this range.

DB2READLOG_READ_SINGLE
Read a single log record (propagatable or not) identified by the starting log sequence number.

DB2READLOG_QUERY
Query the database log. Results of the query will be sent back via the db2ReadLogNoConnInfoStruct structure.

piStartLSN
Input. The starting log sequence number specifies the starting relative byte address for the reading of the log. This value must be the start of an actual log record.

piEndLSN
Input. The ending log sequence number specifies the ending relative byte address for the reading of the log. This value must be greater than *piStartLsn*, and does not need to be the end of an actual log record.

poLogBuffer
Output. The buffer where all the propagatable log records read within the specified range are stored sequentially. This buffer must be large

enough to hold a single log record. As a guideline, this buffer should be a minimum of 32 bytes. Its maximum size is dependent on the size of the requested range. Each log record in the buffer is prefixed by a six byte log sequence number (LSN), representing the LSN of the following log record.

iLogBufferSize
Input. Specifies the size, in bytes, of the log buffer.

piReadLogMemPtr
Input. Block of memory of size iReadLogMemoryLimit that was allocated in the initialization call. This memory contains persistent data that the API requires at each invocation. This memory block must not be reallocated or altered in any way by the caller.

poReadLogInfo
Output. A pointer to the *db2ReadLogNoConnInfoStruct* structure.

firstAvailableLSN
First available LSN in available logs.

firstReadLSN
First LSN read on this call.

nextStartLSN
Next readable LSN.

logRecsWritten
Number of log records written to the log buffer field, *poLogBuffer*.

logBytesWritten
Number of bytes written to the log buffer field, *poLogBuffer*.

lastLogFullyRead
Number indicating the last log file that was read to completion.

Usage notes:

The db2ReadLogNoConn API requires a memory block that must be allocated using the db2ReadLogNoConnInit API. The memory block must be passed as an input parameter to all subsequent db2ReadLogNoConn API calls, and must not be altered.

When requesting a sequential read of log, the API requires a log sequence number (LSN) range and the allocated memory . The API will return a sequence of log records based on the filter option specified when initialized and the LSN range. When requesting a query, the read log information structure will contain a valid starting LSN, to be used on a read call. The value used as the ending LSN on a read can be one of the following:

- A value greater than the caller-specified startLSN.

- FFFF FFFF FFFF which is interpreted by the asynchronous log reader as the end of the available logs.

The propagatable log records read within the starting and ending LSN range are returned in the log buffer. A log record does not contain its LSN, it is contained in the buffer before the actual log record. Descriptions of the various DB2 UDB log records returned by db2ReadLogNoConn can be found in the DB2 UDB Log Records section.

After the initial read, in order to read the next sequential log record, use the nextStartLSN value returned in db2ReadLogNoConnInfoStruct. Resubmit the call, with this new starting LSN and a valid ending LSN and the next block of records is then read. An sqlca code of SQLU_RLOG_READ_TO_CURRENT means the log reader has read to the end of the available log files.

When the API will no longer be used, use db2ReadLogNoConnTerm to terminate the memory.

Related reference:
- "SQLCA" on page 478
- "db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection" on page 192
- "db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection" on page 195

db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection

Allocates the memory to be used by db2ReadLogNoConn in order to extract log records from the DB2 UDB database logs and query the Log Manager for current log state information.

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnInit */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnInit (
  db2Uint32 versionNumber,
  void * pDB2ReadLogNoConnInitStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnInitStruct
{
  db2Uint32                   iFilterOption;
  char                        *piLogFilePath;
  char                        *piOverflowLogPath;
  db2Uint32                   iRetrieveLogs;
  char                        *piDatabaseName;
  char                        *piNodeName;
  db2Uint32                   iReadLogMemoryLimit;
  char                        **poReadLogMemPtr;
} db2ReadLogNoConnInitStruct;
/* ... */
```

API parameters:

version
: Input. Specifies the version and release level of the structure passed as the second parameter *pDB2ReadLogNoConnInitStruct*.

pDB2ReadLogNoConnInitStruct
: Input. A pointer to the *db2ReadLogNoConnInitStruct* structure.

pSqlca
: Output. A pointer to the *sqlca* structure.

iFilterOption
: Input. Specifies the level of log record filtering to be used when reading the log records. Valid values are:

  DB2READLOG_FILTER_OFF
  : Read all log records in the given LSN range.

  DB2READLOG_FILTER_ON
  : Reads only log records in the given LSN range marked as propagatable. This is the traditional behavior of the asynchronous log read API.

piLogFilePath
: Input. Path where the log files to be read are located.

piOverflowLogPath
: Input. Alternate path where the log files to be read may be located.

iRetrieveLogs
: Input. Option specifying if userexit should be invoked to retrieve log files that cannot be found in either the log file path or the overflow log path. Valid values are:

  DB2READLOG_RETRIEVE_OFF
  : Userexit should not be invoked to retrieve missing log files.

DB2READLOG_RETRIEVE_LOGPATH
Userexit should be invoked to retrieve missing log files into the specified log file path.

DB2READLOG_RETRIEVE_OVERFLOW
Userexit should be invoked to retrieve missing log files into the specified overflow log path.

piDatabaseName
Input. Name of the database that owns the recovery logs being read. This is required if the retrieve option above is specified.

piNodeName
Input. Name of the node that owns the recovery logs being read. This is required if the retrieve option above is specified.

iReadLogMemoryLimit
Input. Maximum number of bytes that the API may allocate internally.

poReadLogMemPtr
Output. API-allocated block of memory of size *iReadLogMemoryLimit*. This memory contains persistent data that the API requires at each invocation. This memory block must not be reallocated or altered in any way by the caller.

Usage notes:

The memory initialized by db2ReadLogNoConnInit must not be altered.

When db2ReadLogNoConn will no longer be used, invoke db2ReadLogNoConnTerm to deallocate the memory initialized by db2ReadLogNoConnInit.

Related reference:
- "SQLCA" on page 478
- "db2ReadLogNoConn - Read Log Without a Database Connection" on page 189
- "db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection" on page 195

db2ReadLogNoConnTerm - Terminate Read Log Without a Database Connection

Deallocates the memory used by db2ReadLogNoConn, originally initialized by db2ReadLogNoConnInit.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnTerm */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnTerm (
  db2Uint32 versionNumber,
  void * pDB2ReadLogNoConnTermStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
  char                        **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed as the second parameter *pDB2ReadLogNoConnTermStruct*.

pDB2ReadLogNoConnTermStruct
Input. A pointer to the *db2ReadLogNoConnTermStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

poReadLogMemPtr
Output. Pointer to the block of memory allocated in the initialization call. This pointer will be freed and set to NULL.

Related reference:

- "SQLCA" on page 478
- "db2ReadLogNoConn - Read Log Without a Database Connection" on page 189
- "db2ReadLogNoConnInit - Initialize Read Log Without a Database Connection" on page 192

db2Reorg - Reorganize

Reorganize a table or all indexes defined on a table by compacting the information and reconstructing the rows or index data to eliminate fragmented data.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Reorg */
/* ... */
SQL_API_RC SQL_API_FN
  db2Reorg (
    db2Uint32 versionNumber,
    void *pReorgStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2ReorgStruct
```

```
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2ReorgObject reorgObject;
} db2ReorgStruct;

union db2ReorgObject
{
  struct db2ReorgTable tableStruct;
  struct db2ReorgIndexesAll indexesAllStruct;
};

typedef SQL_STRUCTURE db2ReorgTable
{
  char *pTableName;
  char *pOrderByIndex;
  char *pSysTempSpace;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
  char *pTableName;
} db2ReorgIndexesAll;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gReorg */
/* ... */
SQL_API_RC SQL_API_FN
  db2gReorg (
    db2Uint32 versionNumber,
    void *pReorgStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2gReorgObject reorgObject;
} db2gReorgStruct;

typedef SQL_STRUCTURE db2gReorgNodes
{
  SQL_PDB_NODE_TYPE  nodeNum[SQL_PDB_MAX_NUM_NODE];
} db2gReorgNodes;

union db2gReorgObject
```

```
{
  struct db2gReorgTable tableStruct;
  struct db2gReorgIndexesAll indexesAllStruct;
};

typedef SQL_STRUCTURE db2gReorgTable
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 orderByIndexLen;
  char *pOrderByIndex;
  db2Uint32 sysTempSpaceLen;
  char *pSysTempSpace;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
  db2Uint32  tableNameLen;
  char       *pTableName;
} db2gReorgIndexesAll;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter, *pReorgStruct*.

pReorgStruct
Input. A pointer to the *db2ReorgStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

reorgType
Input. Specifies the type of reorganization. Valid values (defined in `db2ApiDf.h`) are:

DB2REORG_OBJ_TABLE_OFFLINE
Reorganize the table offline.

DB2REORG_OBJ_TABLE_INPLACE
Reorganize the table inplace.

DB2REORG_OBJ_INDEXESALL
Reorganize all indexes.

reorgFlags
Input. Reorganization options. Valid values (defined in `db2ApiDf.h`) are:

DB2REORG_OPTION_NONE
Default action.

DB2REORG_LONGLOB
Reorganize long fields and lobs, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the *reorgType*.

DB2REORG_INDEXSCAN
Recluster utilizing index scan, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the *reorgType*.

DB2REORG_START_ONLINE
Start online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the *reorgType*.

DB2REORG_PAUSE_ONLINE
Pause an existing online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the *reorgType*.

DB2REORG_STOP_ONLINE
Stop an existing online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the *reorgType*.

DB2REORG_RESUME_ONLINE
Resume a paused online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the *reorgType*.

DB2REORG_NOTRUNCATE_ONLINE
Do not perform table truncation, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the *reorgType*.

DB2REORG_ALLOW_NONE
No read or write access to the table. This parameter is not supported when DB2REORG_OBJ_TABLE_INPLACE is specified as the *reorgType*.

DB2REORG_ALLOW_WRITE
Allow read and write access to the table. This parameter is not supported when DB2REORG_OBJ_TABLE_OFFLINE is specified as the *reorgType*.

DB2REORG_ALLOW_READ
Allow only read access to the table.

DB2REORG_CLEANUP_NONE
No clean up is required, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

DB2REORG_CLEANUP_ALL
Clean up the indexes on a table by removing the committed pseudo deleted keys and committed pseudo empty pages, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

DB2REORG_CLEANUP_PAGES
Clean up committed pseudo empty pages only, but do not clean up pseudo deleted keys on pages that are not pseudo empty, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

DB2REORG_CONVERT_NONE
No conversion is required, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

DB2REORG_CONVERT
Convert to type 2 index, used when DB2REORG_OBJ_INDEXESALL is specified as the *reorgType*.

nodeListFlag
Input. Specifies which nodes to reorganize. Valid values (defined in `db2ApiDf.h`) are:

DB2REORG_NODE_LIST
Submit to all nodes in the nodelist array.

DB2REORG_ALL_NODES
Submit to all nodes in the database partition group.

DB2REORG_ALL_EXCEPT
Submit to all nodes except the ones specified by the nodelist parameter.

numNodes
Input. Number of nodes in the nodelist array.

pNodeList
A pointer to the array of node numbers.

reorgObject
Input. Specifies the type of object to be reorganized.

tableStruct
Specifies the options for a table reorganization.

indexesAllStruct
Specifies the options for an index reorganization.

tableNameLen
Input. Specifies the length in bytes of pTableName.

pTableName
Input. Specifies the name of the object to reorganize.

orderByIndexLen
Input. Specifies the length in byte of pOrderByIndex.

pOrderByIndex
Input. Specifies the index to order the table by.

sysTempSpaceLen
Input. Specifies the length in bytes of pSysTempSpace.

pSysTempSpace
Input. Specifies the system temporary table space to create temporary object in.

Usage notes:

Performance of table access, index scans, and the effectiveness of index page prefetching can be adversely affected when the table data has been modified many times, becoming fragmented and unclustered. Use REORGCHK to determine whether a table or its indexes are candidates for reorganizing. All work will be committed and all open cursors will be closed. After reorganizing a table or its indexes, use db2Runstats to update the statistics and sqlarbnd to rebind the packages that use this table.

If the table is partitioned onto several nodes and the reorganization fails on any of the affected nodes, then only the failing nodes will have the table reorganization rolled back.

Note: If table reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.

If the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is *not* specified, and if a clustering index exists, the data will be ordered according to the clustering index.

The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.

To complete a table space roll-forward recovery following a table reorganization, both data and LONG table spaces must be roll-forward enabled.

If the table contains LOB columns that do not use the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS/DMS).

When reorganizing indexes, use the access option to allow other transactions either read only or read-write access to the table. There is a brief lock-out period when the reorganized indexes are being made available during which no access to the table is allowed.

If an index reorganization with allow read or allow write access fails because the indexes need to be rebuilt, the reorganization will be switched to allow no access and carry on. A message will be written to both the administration notification log and the diagnostics log to warn the user about the change in access mode.

If an index reorganization with no access fails, the indexes are not available and have to be rebuilt on the next table access.

This API cannot be used with:
- views or an index that is based on an index extension
- a DMS table while an online backup of a table space in which the table resides is being performed
- declared temporary tables

Related reference:
- “sqlarbnd - Rebind” on page 262
- “SQLCA” on page 478
- “db2Runstats - Runstats” on page 228

Related samples:
- “dbstat.sqb -- Reorganize table and run statistics (MF COBOL)”
- “tbreorg.sqc -- How to reorganize a table and update its statistics (C)”
- “tbreorg.sqC -- How to reorganize a table and update its statistics (C++)”

db2ResetAlertCfg - Reset Alert Configuration

Resets the health indicator settings for specific objects to the current defaults for that object type or resets the current default health indicator settings for an object type to the install defaults.

Authorization:

One of the following:
- *sysadm*
- *sysmaint*
- *sysctrl*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API:  db2ResetAlertCfg */
/* ... */
SQL_API_RC SQL_API_FN
db2ResetAlertCfg(
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca );

typedef SQL_STRUCTURE db2ResetAlertCfgData
{
  db2Uint32             iObjType;
  char                 *piObjName;
  char                 *piDbname;
} db2ResetAlertCfgData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2ResetAlertCfgData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iObjType
Input. Specifies the type of object for which configuration should be reset. Valid values (defined in `db2ApiDf.h`) are:
- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES

- DB2ALERTCFG_OBJTYPE_TABLESPACES
- DB2ALERTCFG_OBJTYPE_TS_CONTAINERS
- DB2ALERTCFG_OBJTYPE_DATABASE
- DB2ALERTCFG_OBJTYPE_TABLESPACE
- DB2ALERTCFG_OBJTYPE_TS_CONTAINER

piObjName
Input. The name of the table space or table space container when object type, *iObjType*, is set to DB2ALERTCFG_OBJTYPE_TS_CONTAINER or DB2ALERTCFG_OBJTYPE_TABLESPACE. The name of the tablespace container is defined as <tablespace-numericalID>.<tablespace-containter-name>.

piDbname
Input. The alias name for the database for which configuration should be reset when object type, *iObjType*, is set to DB2ALERTCFG_OBJTYPE_TS_CONTAINER, DB2ALERTCFG_OBJTYPE_TABLESPACE, and DB2ALERTCFG_OBJTYPE_DATABASE.

Usage notes:

The current default for the object type is reset when *iObjType* is DB2ALERTCFG_OBJTYPE_DBM, DB2ALERTCFG_OBJTYPE_DATABASES, DB2ALERTCFG_OBJTYPE_TABLESPACES, DB2ALERTCFG_OBJTYPE_TS_CONTAINERS or when *piObjName* and *piDbName* are both NULL. If *iObjType* is DB2ALERTCFG_OBJTYPE_DATABASE, DB2ALERTCFG_OBJTYPE_TABLESPACE, DB2ALERTCFG_OBJTYPE_TS_CONTAINER and *piDbName* and *piObjName* (not needed for database) are specified, then the current settings for that specific object will be reset.

Related reference:
- “SQLCA” on page 478
- “db2GetAlertCfg - Get Alert Configuration” on page 60
- “db2UpdateAlertCfg - Update Alert Configuration” on page 242

db2ResetMonitor - Reset Monitor

Resets the database system monitor data of a specified database, or of all active databases, for the application issuing the call.

Scope:

This API can either affect a given database partition on the instance, or all database partitions on the instance.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

To reset the monitor switches for a remote instance (or a different local instance), it is necessary to first attach to that instance.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2ResetMonitor */
/* ... */
SQL_API_RC SQL_API_FN

db2ResetMonitor (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
        db2Uint32                       iResetAll;
        char                            *piDbAlias;
        db2Uint32                        iVersion;
        db2int32                         iNodeNumber;
}db2ResetMonitorData;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gResetMonitor */
/* ... */
SQL_API_RC SQL_API_FN

db2gResetMonitor (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gResetMonitorData
{
  db2Uint32                   iResetAll;
  char                        *piDbAlias;
  db2Uint32                   iDbAliasLength;
  db2Uint32                   iVersion;
  db2int32                    iNodeNumber;
} db2gResetMonitorData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2ResetMonitorData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iResetAll
Input. The reset flag.

piDbAlias
Input. A pointer to the database alias.

iDbAliasLength
Input. Specifies the length in bytes of *piDbAlias*.

iVersion
Input. Version ID of the database monitor data to collect. The database monitor only returns data that was available for the requested version. Set this parameter to one of the following symbolic constants:

- `SQLM_DBMON_VERSION1`

- `SQLM_DBMON_VERSION2`
- `SQLM_DBMON_VERSION5`
- `SQLM_DBMON_VERSION5_2`
- `SQLM_DBMON_VERSION6`
- `SQLM_DBMON_VERSION7`
- `SQLM_DBMON_VERSION8`

Note: If `SQLM_DBMON_VERSION1` is specified as the version, the APIs cannot be run remotely.

iNodeNumber
Input. The database partition server where the request is to be sent. Based on this value, the request will be processed for the current database partition server, all database partition servers or a user specified database partition server. Valid values are:

- `SQLM_CURRENT_NODE`
- `SQLM_ALL_NODES`
- *`node value`*

Note: For standalone instances `SQLM_CURRENT_NODE` must be used.

Usage notes:

Each process (attachment) has its own private view of the monitor data. If one user resets, or turns off a monitor switch, other users are not affected. When an application first calls any database monitor function, it inherits the default switch settings from the database manager configuration file. These settings can be overridden with db2MonitorSwitches - Get/Update Monitor Switches.

If all active databases are reset, some database manager information is also reset to maintain the consistency of the data that is returned.

This API cannot be used to selectively reset specific data items or specific monitor groups. However, a specific group can be reset by turning its switch off, and then on, using db2MonitorSwitches - Get/Update Monitor Switches.

Related reference:

- "db2GetSnapshot - Get Snapshot" on page 73
- "db2MonitorSwitches - Get/Update Monitor Switches" on page 177
- "db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot Output Buffer" on page 76
- "SQLCA" on page 478

db2Restore - Restore database

Rebuilds a damaged or corrupted database that has been backed up using db2Backup - Backup Database. The restored database is in the same state it was in when the backup copy was made. This utility can also restore to a database with a name different from the database name in the backup image (in addition to being able to restore to a new database).

This utility can also be used to restore DB2 databases created in the two previous releases.

This utility can also restore from a table space level backup.

Scope:

This API only affects the database partition from which it is called.

Authorization:

To restore to an existing database, one of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:
- *sysadm*
- *sysctrl*

Required connection:

Database, to restore to an existing database. This API automatically establishes a connection to the specified database and will release the connection when the restore operation finishes.

Instance and database, to restore to a new database. The instance attachment is required to create the database.

To restore to a new database at an instance different from the current instance (as defined by the value of the DB2INSTANCE environment variable), it is necessary to first attach to the instance where the new database will reside.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Restore */
/* ... */
SQL_API_RC SQL_API_FN
db2Restore (
  db2Uint32      versionNumber,
  void          *pDB2RestoreStruct,
  struct sqlca *pSqlca);
/* ... */

typedef SQL_STRUCTURE db2RestoreStruct
{
  char                        *piSourceDBAlias;
  char                        *piTargetDBAlias;
  char                        oApplicationId[SQLU_APPLID_LEN+1];
  char                        *piTimestamp;
  char                        *piTargetDBPath;
  char                        *piReportFile;
  struct db2TablespaceStruct  *piTablespaceList;
  struct db2MediaListStruct   *piMediaList;
  char                        *piUsername;
  char                        *piPassword;
  char                        *piNewLogPath;
  void                        *piVendorOptions;
  db2Uint32                   iVendorOptionsSize;
  db2Uint32                   iParallelism;
  db2Uint32                   iBufferSize;
  db2Uint32                   iNumBuffers;
  db2Uint32                   iCallerAction;
  db2Uint32                   iOptions;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32                   numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32                   numLocations;
  char                        locationType;
} db2MediaListStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gRestore */
/* ... */
SQL_API_RC SQL_API_FN
db2gRestore (
  db2Uint32      versionNumber,
  void         *pDB2gRestoreStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gRestoreStruct
{
  char                         *piSourceDBAlias;
  db2Uint32                    iSourceDBAliasLen;
  char                         *piTargetDBAlias;
  db2Uint32                    iTargetDBAliasLen;
  char                         *poApplicationId;
  db2Uint32                    iApplicationIdLen;
  char                         *piTimestamp;
  db2Uint32                    iTimestampLen;
  char                         *piTargetDBPath;
  db2Uint32                    iTargetDBPathLen;
  char                         *piReportFile;
  db2Uint32                    iReportFileLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                         *piUsername;
  db2Uint32                    iUsernameLen;
  char                         *piPassword;
  db2Uint32                    iPasswordLen;
  char                         *piNewLogPath;
  db2Uint32                    iNewLogPathLen;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    iParallelism;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iCallerAction;
  db2Uint32                    iOptions;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char               *tablespaces;
  db2Uint32                    numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char               *locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter *pDB2RestoreStruct*.

pDB2RestoreStruct
: Input. A pointer to the *db2RestoreStruct* structure

pSqlca
: Output. A pointer to the *sqlca* structure.

piSourceDBAlias
: Input. A string containing the database alias of the source database backup image.

iSourceDBAliasLen
: Input. A 4-byte unsigned integer representing the length in bytes of the source database alias.

piTargetDBAlias
: Input. A string containing the target database alias. If this parameter is null, the *piSourceDBAlias* will be used.

iTargetDBAliasLen
: Input. A 4-byte unsigned integer representing the length in bytes of the target database alias.

oApplicationId
: Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

poApplicationId
: Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil`). The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

iApplicationIdLen
: Input. A 4-byte unsigned integer representing the length in bytes of the poApplicationId buffer. Should be equal to SQLU_APPLID_LEN+1 (defined in `sqlutil`).

piTimestamp
: Input. A string representing the timestamp of the backup image. This field is optional if there is only one backup image in the source specified.

iTimestampLen
: Input. A 4-byte unsigned integer representing the length in bytes of the piTimestamp buffer.

piTargetDBPath
Input. A string containing the relative or fully qualified name of the target database directory on the server. Used if a new database is to be created for the restored backup; otherwise not used.

piReportFile
Input. The file name, if specified, must be fully qualified. The datalinks files that become unlinked during restore (as a result of a fast reconcile) will be reported.

iReportFileLen
Input. A 4-byte unsigned integer representing the length in bytes of the piReportFile buffer.

piTablespaceList
Input. List of table spaces to be restored. Used when restoring a subset of table spaces from a database or table space backup image. See the *DB2TablespaceStruct* structure . The following restrictions apply:

- The database must be recoverable; that is, log retain or user exits must be enabled.
- The database being restored to must be the same database that was used to create the backup image. That is, table spaces can not be added to a database through the table space restore function.
- The rollforward utility will ensure that table spaces restored in a partitioned database environment are synchronized with any other database partition containing the same table spaces. If a table space restore operation is requested and the *piTablespaceList* is NULL, the restore utility will attempt to restore all of the table spaces in the backup image.

When restoring a table space that has been renamed since it was backed up, the new table space name must be used in the restore command. If the old table space name is used, it will not be found.

piMediaList
Input. Source media for the backup image. The information provided depends on the value of the locationType field. The valid values for locationType (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks, or diskettes).

SQLU_TSM_MEDIA
TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in the locations field.

SQLU_USER_EXIT
User exit. No additional input is required (only available when server is on OS/2).

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

iUsernameLen
Input. A 4-byte unsigned integer representing the length in bytes of piUsername. Set to zero if no user name is provided.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iPasswordLen
Input. A 4-byte unsigned integer representing the length in bytes of piPassword. Set to zero if no password is provided.

piNewLogPath
Input. A string representing the path to be used for logging after the restore has completed. If this field is null the default log path will be used.

iNewLogPathLen
Input. A 4-byte unsigned integer representing the length in bytes of *piNewLogPath*.

piVendorOptions
Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and the code page is not checked for this data.

iVendorOptionsSize
Input. The length of the *piVendorOptions*, which cannot exceed 65535 bytes.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024. The default is 1.

iBufferSize
Input. Backup buffer size in 4KB allocation units (pages). Minimum is

8 units. The default is 1024 units. The size entered for a restore must be equal to or an integer multiple of the buffer size used to produce the backup image.

iNumBuffers
Input. Specifies number of restore buffers to be used.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in `db2ApiDf`) are:

DB2RESTORE_RESTORE
Start the restore operation.

DB2RESTORE_NOINTERRUPT
Start the restore. Specifies that the restore will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the restore have been mounted, and utility prompts are not desired.

DB2RESTORE_CONTINUE
Continue the restore after the user has performed some action requested by the utility (mount a new tape, for example).

DB2RESTORE_TERMINATE
Terminate the restore after the user has failed to perform some action requested by the utility.

DB2RESTORE_DEVICE_TERMINATE
Remove a particular device from the list of devices used by restore. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action to remove the device which generated the warning from the list of devices being used.

DB2RESTORE_PARM_CHK
Used to validate parameters without performing a restore. This option does not terminate the database connection after the call returns. After successful return of this call, it is expected that the user will issue a call with DB2RESTORE_CONTINUE to proceed with the action.

DB2RESTORE_PARM_CHK_ONLY
Used to validate parameters without performing a restore. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

DB2RESTORE_TERMINATE_INCRE
Terminate an incremental restore operation before completion.

DB2RESTORE_RESTORE_STORDEF
Initial call. Table space container redefinition requested.

DB2RESTORE_STORDEF_NOINTERRUPT
Initial call. The restore will run uninterrupted. Table space container redefinition requested.

iOptions
Input. A bitmap of restore properties. The options are to be combined using the bitwise OR operator to produce a value for *iOptions*. Valid values (defined in `db2ApiDf`) are:

DB2RESTORE_OFFLINE
Perform an offline restore operation.

DB2RESTORE_ONLINE
Perform an online restore operation.

DB2RESTORE_DB
Restore all table spaces in the database. This must be run offline

DB2RESTORE_TABLESPACE
Restore only the table spaces listed in the *piTablespaceList* parameter from the backup image. This can be online or offline.

DB2RESTORE_HISTORY
Restore only the history file.

DB2RESTORE_INCREMENTAL
Perform a manual cumulative restore operation.

DB2RESTORE_AUTOMATIC
Perform an automatic cumulative (incremental) restore operation. Must be specified with DB2RESTORE_INCREMENTAL.

DB2RESTORE_DATALINK
Perform reconciliation operations. Tables with a defined DATALINK column must have RECOVERY YES option specified.

DB2RESTORE_NODATALINK
Do not perform reconciliation operations. Tables with DATALINK columns are placed into DataLink_Roconcile_pending (DRP) state. Tables with a defined DATALINK column must have the RECOVERY YES option specified.

DB2RESTORE_ROLLFWD
Place the database in rollforward pending state after it has been successfully restored.

DB2RESTORE_NOROLLFWD
Do not place the database in rollforward pending state after it has been successfully restored. This cannot be specified for backups taken online or for table space level restores. If, following a successful restore, the database is in roll-forward pending state, db2Rollforward - Rollforward Database must be executed before the database can be used.

tablespaces
A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of *db2Char* structures.

numTablespaces
Number of entries in the *tablespaces* parameter.

locations
A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
The number of entries in the *locations* parameter.

locationType
A character indicated the media type. Valid values (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices(tapes, disks, diskettes, or named pipes).

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

pioData
A pointer to the character data buffer.

iLength
Input. The size of the *pioData* buffer

oLength
Output. Reserved for future use.

Usage notes:

For offline restore, this utility connects to the database in exclusive mode. The utility fails if any application, including the calling application, is already connected to the database that is being restored. In addition, the request will fail if the restore utility is being used to perform the restore, and any application, including the calling application, is already connected to any database on the same workstation. If the connect is successful, the API locks out other applications until the restore is completed.

The current database configuration file will not be replaced by the backup copy unless it is unusable. If the file is replaced, a warning message is returned.

The database or table space must have been backed up using db2Backup - Backup Database.

If the caller action is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action is DB2RESTORE_RESTORE, and the utility is restoring to an existing database, the utility returns control to the application with a message requesting some user interaction. After handling the user interaction, the application calls RESTORE DATABASE again, with the caller action set to indicate whether processing is to continue (DB2RESTORE_CONTINUE) or terminate (DB2RESTORE_TERMINATE) on the subsequent call. The utility finishes processing, and returns an SQLCODE in the *sqlca*.

To close a device when finished, set the caller action to DB2RESTORE_DEVICE_TERMINATE. If, for example, a user is restoring from 3 tape volumes using 2 tape devices, and one of the tapes has been restored, the application obtains control from the API with an SQLCODE indicating end of tape. The application can prompt the user to mount another tape, and if the user indicates "no more", return to the API with caller action SQLUD_DEVICE_TERMINATE to signal end of the media device. The device driver will be terminated, but the rest of the devices involved in the restore will continue to have their input processed until all segments of the restore set have been restored (the number of segments in the restore set is placed on the last media device during the backup process). This caller action can be used with devices other than tape (vendor supported devices).

To perform a parameter check before returning to the application, set caller action to DB2RESTORE_PARM_CHK.

Set caller action to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with sqlbstsc - Set Tablespace Containers.

If a system failure occurs during a critical stage of restoring a database, the user will not be able to successfully connect to the database until a successful restore is performed. This condition will be detected when the connection is attempted, and an error message is returned. If the backed-up database is not configured for roll-forward recovery, and there is a usable current configuration file with either of these parameters enabled, following the restore, the user will be required to either take a new backup of the database, or disable the log retain and user exit parameters before connecting to the database.

Although the restored database will not be dropped (unless restoring to a nonexistent database), if the restore fails, it will not be usable.

If the restore type specifies that the history file on the backup is to be restored, it will be restored over the existing history file for the database, effectively erasing any changes made to the history file after the backup that is being restored. If this is undesirable, restore the history file to a new or test database so that its contents can be viewed without destroying any updates that have taken place.

If, at the time of the backup operation, the database was enabled for roll forward recovery, the database can be brought to the state it was in prior to the occurrence of the damage or corruption by issuing db2Rollforward after successful execution of db2Restore. If the database is recoverable, it will default to roll forward pending state after the completion of the restore.

If the database backup image is taken offline, and the caller does not want to roll forward the database after the restore, the DB2RESTORE_NOROLLFWD option can be used for the restore. This results in the database being useable immediately after the restore. If the backup image is taken online, the caller must roll forward through the corresponding log records at the completion of the restore.

Related reference:
- "sqlemgdb - Migrate Database" on page 368
- "db2Rollforward - Rollforward Database" on page 219
- "SQLCA" on page 478
- "db2Backup - Backup database" on page 31
- "db2CfgGet - Get Configuration Parameters" on page 39

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2Rollforward - Rollforward Database

Recovers a database by applying transactions recorded in the database log files. Called after a database or a table space backup has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either *logretain*, *userexit*, or both of these database configuration parameters must be set on) before the database can be recovered with roll-forward recovery.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. A database or table space rollforward call specifying a point-in-time affects all database partition servers that are listed in the `db2nodes.cfg` file. A database or table space rollforward call specifying end of logs affects the database partition servers that are specified. If no database partition servers are specified, it affects all database partition servers that are listed in the `db2nodes.cfg` file; if no roll forward is needed on a particular database partition server, that database partition server is ignored.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None. This API establishes a database connection.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2Rollforward_api (
  db2Uint32 versionNumber,
  void *pDB2RollforwardStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
```

```
{
  struct db2RfwdInputStruct   *roll_input;
  struct db2RfwdOutputStruct  *roll_output;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
  sqluint32                   version;
  char                        *pDbAlias;
  db2Uint32                   CallerAction;
  char                        *pStopTime;
  char                        *pUserName;
  char                        *pPassword;
  char                        *pOverflowLogPath;
  db2Uint32                   NumChngLgOvrflw;
  struct sqlurf_newlogpath    *pChngLogOvrflw;
  db2Uint32                   ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  db2int32                    AllNodeFlag;
  db2int32                    NumNodes;
  SQL_PDB_NODE_TYPE           *pNodeList;
  db2int32                    NumNodeInfo;
  char                        *pDroppedTblID;
  char                        *pExportDir;
  db2Uint32                   RollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *pApplicationId;
  sqlint32                    *pNumReplies;
  struct sqlurf_info          *pNodeInfo;
} db2RfwdOutputStruct;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2gRollforward_api (
  db2Uint32 versionNumber,
  void *pDB2gRollforwardStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct  *roll_input;
  struct db2RfwdOutputStruct  *roll_output;
} db2gRollforwardStruct;

SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32                   DbAliasLen;
  db2Uint32                   StopTimeLen;
  db2Uint32                   UserNameLen;
  db2Uint32                   PasswordLen;
  db2Uint32                   OvrflwLogPathLen;
  db2Uint32                   DroppedTblIDLen;
  db2Uint32                   ExportDirLen;
  sqluint32                   Version;
  char                        *pDbAlias;
  db2Uint32                   CallerAction;
  char                        *pStopTime;
  char                        *pUserName;
  char                        *pPassword;
  char                        *pOverflowLogPath;
  db2Uint32                   NumChngLgOvrflw;
  struct sqlurf_newlogpath    *pChngLogOvrflw;
  db2Uint32                   ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  db2int32                    AllNodeFlag;
  db2int32                    NumNodes;
  SQL_PDB_NODE_TYPE           *pNodeList;
  db2int32                    NumNodeInfo;
  char                        *pDroppedTblID;
  char                        *pExportDir;
  db2Uint32                   RollforwardFlags;
};

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char                        *pApplicationId;
  sqlint32                    *pNumReplies;
  struct sqlurf_info          *pNodeInfo;
} db2RfwdOutputStruct;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter.

pDB2RollforwardStruct
Input. A pointer to the *db2RollforwardStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

roll_input
Input. A pointer to the *db2RfwdInputStruct* structure.

roll_output
Output. A pointer to the *db2RfwdOutputStruct* structure.

DbAliasLen
Input. Specifies the length in bytes of the database alias.

StopTimeLen
Input. Specifies the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

UserNameLen
Input. Specifies the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
Input. Specifies the length in bytes of the password. Set to zero if no password is provided.

OverflowLogPathLen
Input. Specifies the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

Version
Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

pDbAlias
Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

CallerAction
Input. Specifies action to be taken. Valid values (defined in `sqlutil`) are:

SQLUM_ROLLFWD
Rollforward to the point in time specified by *pPointInTime*. For database rollforward, the database is left in *rollforward-pending* state. For table space rollforward to a point in time, the table spaces are left in *rollforward-in-progress* state.

SQLUM_STOP
End roll-forward recovery. No new log records are processed and uncommitted transactions are backed out. The *rollforward-pending* state of the database or table spaces is turned off. Synonym is SQLUM_COMPLETE.

SQLUM_ROLLFWD_STOP
Rollforward to the point in time specified by *pPointInTime*, and end roll-forward recovery. The *rollforward-pending* state of the database or table spaces is turned off. Synonym is SQLUM_ROLLFWD_COMPLETE.

SQLUM_QUERY
Query values for *pNextArcFileName*, *pFirstDelArcFileName*, *pLastDelArcFileName*, and *pLastCommitTime*. Return database status and a node number.

SQLUM_PARM_CHECK
: Validate parameters without performing the roll forward.

SQLUM_CANCEL
: Cancel the rollforward operation that is currently running. The database or table space are put in recovery pending state.

  Note: This option cannot be used while the rollforward is actually running. It can be used if the rollforward is paused (that is, waiting for a STOP), or if a system failure occurred during the rollforward. It should be used with caution.

Rolling databases forward may require a load recovery using tape devices. The rollforward API will return with a warning message if user intervention on a device is required. The API can be called again with one of the following three caller actions:

SQLUM_LOADREC_CONTINUE
: Continue using the device that generated the warning message (for example, when a new tape has been mounted).

SQLUM_LOADREC_DEVICE_TERMINATE
: Stop using the device that generated the warning message (for example, when there are no more tapes).

SQLUM_LOADREC_TERMINATE
: Terminate all devices being used by load recovery.

pStopTime
: Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for `SQLUM_QUERY`, `SQLUM_PARM_CHECK`, and any of the load recovery (`SQLUM_LOADREC_`*xxx*) caller actions.

pUserName
: Input. A string containing the user name of the application. May be NULL.

pPassword
: Input. A string containing the password of the supplied user name (if any). May be NULL.

pOverflowLogPath
: Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the *logpath* before they can be used by this utility. This can be a problem if the user does not have sufficient space in the *logpath*. The overflow log path is provided for this reason.

During roll-forward recovery, the required log files are searched, first in the *logpath*, and then in the overflow log path. The log files needed for table space roll-forward recovery can be brought into either the *logpath* or the overflow log path. If the caller does not specify an overflow log path, the default value is the *logpath*. In a partitioned database environment, the overflow log path must be a valid, fully qualified path; the default path is the default overflow log path for each node. In a single-partition database environment, the overflow log path can be relative if the server is local.

NumChngLgOvrflw
Partitioned database environments only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

pChngLogOvrflw
Partitioned database environments only. A pointer to a structure containing the fully qualified names of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

ConnectMode
Input. Valid values (defined in `sqlutil`) are:

SQLUM_OFFLINE
Offline roll forward. This value must be specified for database roll-forward recovery.

SQLUM_ONLINE
Online roll forward.

pTablespaceList
Input. A pointer to a structure containing the names of the table spaces to be rolled forward to the end-of-logs or to a specific point in time. If not specified, the table spaces needing rollforward will be selected.

AllNodeFlag
Partitioned database environments only. Input. Indicates whether the rollforward operation is to be applied to all database partition servers defined in `db2nodes.cfg`. Valid values are:

SQLURF_NODE_LIST
Apply to database partition servers in a list that is passed in *pNodeList*.

SQLURF_ALL_NODES
Apply to all database partition servers. *pNodeList* should be NULL. This is the default value.

SQLURF_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in *pNodeList*.

SQLURF_CAT_NODE_ONLY
Apply to the catalog partition only. *pNodeList* should be NULL.

NumNodes
Input. Specifies the number of database partition servers in the *pNodeList* array.

pNodeList
Input. A pointer to an array of database partition server numbers on which to perform the roll-forward recovery.

NumNodeInfo
Input. Defines the size of the output parameter *pNodeInfo*, which must be large enough to hold status information from each database partition that is being rolled forward. In a single-partition database environment, this parameter should be set to 1. The value of this parameter should be same as the number of database partition servers for which this API is being called.

pDroppedTblID
Input. A string containing the ID of the dropped table whose recovery is being attempted.

pExportDir
Input. The directory into which the dropped table data will be exported.

RollforwardFlags
Input. Specifies the rollforward flags. Valid values (defined in `sqlpapiRollforward`):

SQLP_ROLLFORWARD_LOCAL_TIME
Allows the user to rollforward to a point in time that is the user's local time rather than GMT time. This makes it easier for users to rollforward to a specific point in time on their local machines, and eliminates potential user errors due to the translation of local to GMT time.

SQLP_ROLLFORWARD_NO_RETRIEVE
Controls which log files to be rolled forward on the standby machine by allowing the user to disable the retrieval of archived logs. By controlling the log files to be rolled forward, one can ensure that the standby machine is `X` hours behind the production machine, to prevent the user affecting both systems. This option is useful if the standby system does not

have access to archive, for example, if TSM is the archive, it only allows the original machine to retrieve the files. It will also remove the possibility that the standby system would retrieve an incomplete log file while the production system is archiving a file and the standby system is retrieving the same file.

pApplicationId
Output. The application ID.

pNumReplies
Output. The number of replies received.

pNodeInfo
Output. Database partition reply information.

Usage notes:

The database manager uses the information stored in the archived and the active log files to reconstruct the transactions performed on the database since its last backup.

The action performed when this API is called depends on the *rollforward_pending* flag of the database prior to the call. This can be queried using db2CfgGet - Get Configuration Parameters The *rollforward_pending* flag is set to `DATABASE` if the database is in roll-forward pending state. It is set to `TABLESPACE` if one or more table spaces are in `SQLB_ROLLFORWARD_PENDING` or `SQLB_ROLLFORWARD_IN_PROGRESS` state. The *rollforward_pending* flag is set to `NO` if neither the database nor any of the table spaces needs to be rolled forward.

If the database is in roll-forward pending state when this API is called, the database will be rolled forward. Table spaces are returned to normal state after a successful database roll-forward, unless an abnormal state causes one or more table spaces to go offline. If the *rollforward_pending* flag is set to `TABLESPACE`, only those table spaces that are in roll-forward pending state, or those table spaces requested by name, will be rolled forward.

Note: If table space rollforward terminates abnormally, table spaces that were being rolled forward will be put in `SQLB_ROLLFORWARD_IN_PROGRESS` state. In the next invocation of ROLLFORWARD DATABASE, only those table spaces in `SQLB_ROLLFORWARD_IN_PROGRESS` state will be processed. If the set of selected table space names does not include all table spaces that are in `SQLB_ROLLFORWARD_IN_PROGRESS` state, the table spaces that are not required will be put into `SQLB_RESTORE_PENDING` state.

If the database is not in roll-forward pending state and no point in time is specified, any table spaces that are in rollforward-in-progress state will be rolled forward to the end of logs. If no table spaces are in rollforward-in-progress state, any table spaces that are in rollforward pending state will be rolled forward to the end of logs.

This API reads the log files, beginning with the log file that is matched with the backup image. The name of this log file can be determined by calling this API with a caller action of `SQLUM_QUERY` before rolling forward any log files.

The transactions contained in the log files are reapplied to the database. The log is processed as far forward in time as information is available, or until the time specified by the stop time parameter.

Recovery stops when any one of the following events occurs:
- No more log files are found
- A time stamp in the log file exceeds the completion time stamp specified by the stop time parameter
- An error occurs while reading the log file.

Some transactions might not be recovered. The value returned in *pLastCommitTime* indicates the time stamp of the last committed transaction that was applied to the database.

If the need for database recovery was caused by application or human error, the user may want to provide a time stamp value in *pStopTime*, indicating that recovery should be stopped before the time of the error. This applies only to full database roll-forward recovery, and to table space rollforward to a point in time. It also permits recovery to be stopped before a log read error occurs, determined during an earlier failed attempt to recover.

When the *rollforward_recovery* flag is set to `DATABASE`, the database is not available for use until roll-forward recovery is terminated. Termination is accomplished by calling the API with a caller action of `SQLUM_STOP` or `SQLUM_ROLLFORWARD_STOP` to bring the database out of roll-forward pending state. If the *rollforward_recovery* flag is `TABLESPACE`, the database is available for use. However, the table spaces in `SQLB_ROLLFORWARD_PENDING` and `SQLB_ROLLFORWARD_IN_PROGRESS` states will not be available until the API is called to perform table space roll-forward recovery. If rolling forward table spaces to a point in time, the table spaces are placed in backup pending state after a successful rollforward.

When the *RollforwardFlags* option is set to `SQLP_ROLLFORWARD_LOCAL_TIME`, all messages returned to the user will also be in local time. All times are converted on the server, and on the catalog partition, if it is a partitioned database environment. The timestamp string is converted to GMT on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.

Related reference:

- "SQLCA" on page 478
- "db2Restore - Restore database" on page 208

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2Runstats - Runstats

Updates statistics about the characteristics of a table and/or any associated indexes. These characteristics include, among many others, number of records, number of pages, and average record length. The optimizer uses these statistics when determining access paths to the data.

This utility should be called when a table has had many updates, after reorganizing a table, or after creating a new index.

Statistics are collected based on the table partition that is resident on the database partition where the API executes. Global table statistics are derived by multiplying the values obtained at a database partition by the number of database partitions on which the table is completely stored. The global statistics are stored in the catalog tables.

The database partition from which the API is called does not have to contain a partition for the table:

- If the API is called from a database partition that contains a partition for the table, the utility executes at this database partition.
- If the API is called from a database partition that does not contain a table partition, the request is sent to the first database partition in the database partition group that holds a partition for the table. The utility then executes at this database partition.

Scope:

This API can be called from any database partition server in the `db2nodes.cfg` file. It can be used to update the catalogs on the catalog database partition.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- CONTROL privilege on the table
- LOAD

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2Runstats */
/* ... */
SQL_API_RC  SQL_API_FN
   db2Runstats (
    db2Uint32 versionNumber,
    db2RunstatsData *data,
    struct sqlca *sqlca);

typedef SQL_STRUCTURE db2RunstatsData
{
  double                       iSamplingOption;
  unsigned char                *piTablename;
  db2ColumnData                **piColumnList;
  db2ColumnDistData            **piColumnDistributionList;
  db2ColumnGrpData             **piColumnGroupList;
  unsigned char                **piIndexList;
  db2Uint32                    iRunstatsFlags;
  db2int16                     iNumColumns;
  db2int16                     iNumColdist;
  db2int16                     iNumColGroups;
  db2int16                     iNumIndexes;
  db2int16                     iParallelismOption;
  db2int16                     iTableDefaultFreqValues;
  db2int16                     iTableDefaultQuantiles;
} db2RunstatsData;

typedef SQL_STRUCTURE db2ColumnData
```

```
{
  unsigned char              *piColumnName;
  db2int16                    iColumnFlags;
} db2ColumnData;

typedef SQL_STRUCTURE db2ColumnDistData
{
  unsigned char              *piColumnName;
  db2int16                   iNumFreqValues;
  db2int16                   iNumQuantiles;
} db2ColumnDistData;

typedef SQL_STRUCTURE db2ColumnGrpData
{
  unsigned char              **piGroupColumnNames;
  db2int16                   iGroupSize;
  db2int16                   iNumFreqValues;
  db2int16                   iNumQuantiles;
} db2ColumnGrpData;
/* ... */
```

Generic API syntax:

```
/* File: db2ApiDf.h */
/* API: db2gRunstats */
/* ... */
SQL_API_RC  SQL_API_FN
   db2gRunstats (
    db2Uint32 versionNumber,
    db2gRunstatsData *data,
    struct sqlca *sqlca);

typedef SQL_STRUCTURE db2gRunstatsData
{
  double                     iSamplingOption;
  unsigned char              *piTablename;
  db2gColumnData             **piColumnList;
  db2gColumnDistData         **piColumnDistributionList;
  db2gColumnGrpData          **piColumnGroupList;
  unsigned char              **piIndexList;
  db2Uint16                  *piIndexNamesLen;
  db2Uint32                  iRunstatsFlags;
  db2Uint16                  iTablenameLen;
  db2int16                   iNumColumns;
  db2int16                   iNumColdist;
  db2int16                   iNumColGroups;
  db2int16                   iNumIndexes;
  db2int16                   iParallelismOption;
  db2int16                   iTableDefaultFreqValues;
  db2int16                   iTableDefaultQuantiles;
} db2gRunstatsData;

typedef SQL_STRUCTURE db2gColumnData
{
  unsigned char              *piColumnName;
  db2Uint16                  iColumnNameLen;
  db2int16                   iColumnFlags;
} db2gColumnData;

typedef SQL_STRUCTURE db2gColumnDistData
{
  unsigned char              *piColumnName;
  db2Uint16                  iColumnNameLen;
  db2int16                   iNumFreqValues;
  db2int16                   iNumQuantiles;
} db2gColumnDistData;

typedef SQL_STRUCTURE db2gColumnGrpData
{
  unsigned char              **piGroupColumnNames;
  db2Uint16                  *piGroupColumnNamesLen;
  db2int16                   iGroupSize;
  db2int16                   iNumFreqValues;
  db2int16                   iNumQuantiles;
} db2gColumnGrpData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *data*.

data Input. A pointer to the *db2RunstatsData* structure.

sqlca Output. A pointer to the *sqlca* structure.

iSamplingOption
Input. Reserved for future use. Valid values are 0 or 100.

piTablename
Input. A pointer to the fully qualified name of the table on which statistics are to be gathered. The name can be an alias. For row types, *piTablename* must be the name of the hierarchy's root table.

piColumnList
Input. An array of *db2ColumnData* elements. Each element of this array is made up of two sub-elements:

- a string that represents the name of the column on which to collect statistics
- a flags field indicating statistic options for the column

If *iNumColumns* is zero then *piColumnList* is ignored if provided.

piColumnDistributionList
Input. An array of *db2ColumnDistData* elements. These elements are

provided when collecting distribution statistics on a particular column or columns is desired. Each element of this array is made up of three sub-elements :

- a string that represents the name of the column on which to collect distribution statistics
- the number of frequent values to collect.
- the number of quantiles to collect.

Any columns which appear in the *piColumnDistributionList* that do NOT appear in the *piColumnList*, will have basic column statistics collected on them. This would be the same effect as having included these columns in the *piColumnList* in the first place. If *iNumColdist* is zero then *piColumnDistributionList* is ignored.

piColumnGroupList

Input. An array of db2ColumnGrpData elements. These elements are provided when collecting column statistics on a group of columns. That is, the values in each column of the group for each row will be concatenated together and treated as a single value. Each db2ColumnGrpData is made up of 3 integer fields and an array of strings. The first integer field represents the number of strings in the array of strings piGroupColumns. Each string in this array contains one column name. For example, if column combinations statistics are to be collected on column groups (c1,c2) and on (c3,c4,c5) then there are 2 db2ColumnGrpData elements in piGroupColumns.

The first db2ColumnGrpData element is as follows: piGroupSize = 2 and the array of strings contains 2 elements, namely, c1 and c2.

The second db2ColumnGrpData element is as follows. piGroupSize = 3 and the array of strings contains 3 elements, namely, c3, c4 and c5.

The second and the third integer fields represent the number of frequent values and the number of quantiles respectively when collecting distribution statistics on column groups. This is not currently supported.

Any columns which appear in the *piColumnGroupList* that do NOT appear in the *piColumnList*, will have basic column statistics collected on them. This would be the same effect as having included these columns in the *piColumnList* in the first place. If *iNumColGroups* is zero then *piColumnGroupList* is ignored.

piIndexList

Input. An array of strings. Each string contains one fully qualified index name. If NumIndexes is zero then *piIndexList* is ignored.

piIndexNamesLen
Input. An array of values representing the length in bytes of each of the index names in the index list. If NumIndexes is zero then *piIndexNamesLen* is ignored.

iRunstatsFlags
Input. A bit mask field used to specify statistics options. Valid values are:

DB2RUNSTATS_ALL_COLUMNS
Collect statistics on all columns of the table. This option can be specified in combination with column, column distribution, column group or index structure lists. This is useful if you would like to collect statistics on all columns of the table but would like to provide statistics options for specific columns.

DB2RUNSTATS_KEY_COLUMNS
Collect statistics only on the columns that make up all the indexes defined on the table. This option can be specified in combination with column, column distribution, column group or index structure lists. This is useful if you would like to collect statistics on all key columns of the table but would also like to gather statistics for some non-key columns or would like to provide statistics options for specific key columns.

DB2RUNSTATS_DISTRIBUTION
Collect distribution statistics. This option can only be used with DB2RUNSTATS_ALL_COLUMNS and DB2RUNSTATS_KEY_COLUMNS. When used with DB2RUNSTATS_ALL_COLUMNS, distribution statistics are gathered for all columns of the table. When used with DB2RUNSTATS_KEY_COLUMNS, distribution statistics are gathered for all columns that make up all the indexes defined on the table. When used with both DB2RUNSTATS_ALL_COLUMNS and DB2RUNSTATS_KEY_COLUMNS, basic statistics are gathered for all columns of the table and distribution statistics are gathered for only columns that make up all the indexes defined on the table.

DB2RUNSTATS_ALL_INDEXES
Collect statistics on all indexes defined on the table.

DB2RUNSTATS_EXT_INDEX
Collect detailed index statistics. The option must be specified with either DB2RUNSTATS_ALL_INDEXES or an explicit list of index names (*piIndexList* and *iNumIndexes* > 0).

DB2RUNSTATS_EXT_INDEX_SAMPLED
Collect detailed index statistics using sampling methods. The option must be specified with either DB2RUNSTATS_ALL_INDEXES or an explicit list of index names (*piIndexList* and *iNumIndexes* > 0). DB2RUNSTATS_EXT_INDEX will be ignored if specified at the same time.

DB2RUNSTATS_ALLOW_READ
Allows others to have read-only access while the statistics are being gathered. The default is to allow read and write access.

iTablenameLen
Input. A value representing the length in bytes of the table name.

iNumColumns
Input. The number of items specified in the piColumnList list.

iNumColdist
Input. The number of items specified in the *piColumnDistributionList* list.

iNumColGroups
Input. The number of items specified in the *piColumnGroupList* list.

iNumIndexes
Input. The number of items specified in the *piIndexList* list.

iParallelismOption
Input. Reserved for future use. Valid values are 0.

iTableDefaultFreqValues
Input. Specifies the default number of frequent values to collect for the table. Valid values are:

n n frequent values will be collected unless otherwise specified at the column level.

0 No frequent values will be collected unless otherwise specified at the column level.

-1 Use the default database configuration parameter NUM_FREQVALUES for the number of frequent values to collect.

iTableDefaultQuantiles
Input. Specifies the default number of quantiles to collect for the table. Valid values are:

n n quantiles will be collected unless otherwise specified at the column level.

0 No quantiles will be collected unless otherwise specified at the column level.

-1 Use the default database configuration parameter NUM_QUANTILES for the number of quantiles to collect.

piColumnName
Input. Pointer to a string representing a column name.

iColumnNameLen
Input. A value representing the length in bytes of the column name.

iColumnFlags
Input. A bit mask field used to specify statistics options for the column. Valid values are:

DB2RUNSTATS_COLUMN_LIKE_STATS
Collect LIKE statistics on the column.

iNumFreqValues
Input. The number of frequent values to collect on the column. Valid values are:

n Collect n frequent values on the column.

-1 Use the table default number of frequent values, such as *iTableDefaultFreqValues* if set, or the database configuration parameter NUM_FREQVALUES.

iNumQuantiles
Input. The number of quantiles to collect on the column. Valid values are:

n Collect n quantiles on the column.

-1 Use the table default number of quantiles, *iTableDefaultQuantiles* if set, or the database configuration parameter NUM_QUANTILES.

piGroupColumnNames
Input. An array of strings. Each string represents a column name that is part of the column group on which to collect statistics.

piGroupColumnNamesLen
Input. An array of values representing the length in bytes of each of the column names in the column names list.

iGroupSize
Input. Number of columns in the column group. Valid values are:

n The column group is made up of n columns.

iNumFreqValues
Input. Reserved for future use.

iNumQuantiles
Input. Reserved for future use.

Usage notes:

Use RUNSTATS to update statistics:
- On tables that have been modified many times (for example, if a large number of updates have been made, or if a significant amount of data has been inserted or deleted)
- On tables that have been reorganized
- When a new index has been created.

After statistics have been updated, new access paths to the table can be created by rebinding the packages using sqlabndx - Bind.

If index statistics are requested, and statistics have never been run on the table containing the index, statistics on both the table and indexes are calculated.

After calling this API, the application should issue a COMMIT to release the locks.

To allow new access plans to be generated, the packages that reference the target table must be rebound after calling this API.

Running this API on the table only may result in a situation where the table level statistics are inconsistent with the already existing index level statistics. For example, if index level statistics are collected on a particular table and later a significant number of rows is deleted from this table, issuing this API on the table only may end up with the table cardinality less than FIRSTKEYCARD which is an inconsistent state. Likewise, issuing this API for indexes only may leave the already existing table level statistics in an inconsistent state. For example, if table level statistics are collected on a particular table and later a significant number of rows is deleted from this table, issuing the db2Runstats API for the indexes only may end up with some columns having a COLCARD greater than the table cardinality. A warning will be returned if such an inconsistency is detected.

Related reference:
- "sqlabndx - Bind" on page 253
- "SQLCA" on page 478
- "REORGCHK Command" in the *Command Reference*
- "db2CfgGet - Get Configuration Parameters" on page 39
- "db2Reorg - Reorganize" on page 196

Related samples:

- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"
- "tbreorg.sqc -- How to reorganize a table and update its statistics (C)"
- "tbreorg.sqC -- How to reorganize a table and update its statistics (C++)"

db2SetSyncSession - Set Satellite Sync Session

Sets the synchronization session for a satellite. A synchronization session is associated with the version of the user application executing on the satellite. Each version of an application is supported by a particular database configuration, and manipulates particular data sets, each of which can be synchronized with a central site.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2SetSyncSession */
/* ... */
SQL_API_RC SQL_API_FN
  db2SetSyncSession (
    db2Uint32 versionNumber,
    void *pParmStruct,
    struct sqlca *pSqlca);

typedef struct
{
  char *piSyncSessionID;
} db2SetSyncSessionStruct;
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2SetSyncSessionStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piSyncSessionID
Input. Specifies an identifier for the synchronization session that a satellite will use. The specified value must match the appropriate application version for the satellite's group, as defined at the satellite control server.

Related reference:

- "SQLCA" on page 478

db2SetWriteForDB

Sets the database to be I/O write suspended, or resumes I/O writes to disk. I/O writes must be suspended for a database before a split mirror can be taken. To avoid potential problems, keep the same connection to do the write suspension and resumption.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2SetWriteForDB */
/* ... */
SQL_API_RC SQL_API_FN
db2SetWriteForDB (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
```

```
typedef struct db2SetWriteDbStruct
{
  db2int32                    iOption;
  char                        *piTablespaceNames;
} db2SetWriteDbStruct;
/* ... */
```

API parameters:

version
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2SetWriteDbStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iOption
Input. Specifies the action. Valid values are:

DB2_DB_SUSPEND_WRITE
Suspends I/O write to disk.

DB2_DB_RESUME_WRITE
Resumes I/O write to disk.

piTablespaceNames
Input. Reserved for future use.

db2SyncSatellite - Sync Satellite

Synchronizes a satellite.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

db2SyncSatellite - Sync Satellite

```
/* File: db2ApiDf.h */
/* API: db2SyncSatellite */
/* ... */
SQL_API_RC SQL_API_FN
db2SyncSatellite (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. Set to NULL.

pSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478

db2SyncSatelliteStop - Stop Satellite Sync

Stops the satellite's currently active synchronization session. The session is stopped in such a way that synchronization for this satellite can be restarted where it left off by invoking db2SyncSatellite.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2SyncSatelliteStop */
/* ... */
SQL_API_RC SQL_API_FN
```

```
db2SyncSatelliteStop (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
/* ... */
```

API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
: Input. Set to NULL.

pSqlca
: Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478
- "db2SyncSatellite - Sync Satellite" on page 239

db2SyncSatelliteTest - Test Satellite Sync

Tests the ability of a satellite to synchronize.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2SyncSatelliteTest */
/* ... */
SQL_API_RC SQL_API_FN
db2SyncSatelliteTest (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. Set to NULL.

pSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478

db2UpdateAlertCfg - Update Alert Configuration

Updates the alert configuration settings for health indicators.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2UpdateAlertCfg */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateAlertCfg (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateAlertCfgData
{
   db2Uint32                 iObjType;
   char                      *piObjName;
   char                      *piDbName;
   db2Uint32                 iIndicatorID;
   db2Uint32                 iNumIndAttribUpdates;
   struct db2AlertAttrib     *piIndAttribUpdates;
   db2Uint32                 iNumActionUpdates;
   struct db2AlertActionUpdate *piActionUpdates;
   db2Uint32                 iNumActionDeletes;
   struct db2AlertActionDelete *piActionDeletes;
   db2Uint32                 iNumNewActions;
   struct db2AlertActionNew  *piNewActions;
} db2UpdateAlertCfgData;

typedef SQL_STRUCTURE db2AlertAttrib
{
   db2Uint32                 iAttribID;
   char                      *piAttribValue;
} db2AlertAttrib;

typedef SQL_STRUCTURE db2AlertActionUpdate
{
   db2Uint32                 iActionType;
   char                      *piActionName;
   db2Uint32                 iCondition;
   db2Uint32                 iNumParmUpdates;
   struct db2AlertAttrib     *piParmUpdates;
} db2AlertActionUpdate;

typedef SQL_STRUCTURE db2AlertActionDelete
{
   db2Uint32                 iActionType;
   char                      *piName;
   db2Uint32                 iCondition;
} db2AlertActionDelete;

typedef SQL_STRUCTURE db2AlertActionNew
{
   db2Uint32                 iActionType;
   struct db2AlertScriptAction *piScriptAttribs;
   struct db2AlertTaskAction *piTaskAttribs;
} db2AlertActionNew;

typedef SQL_STRUCTURE db2AlertScriptAction
{
   db2Uint32                 scriptType;
   db2Uint32                 condition;
   char                      *pPathName;
   char                      *pWorkingDir;
   char                      *pCmdLineParms;
   char                      stmtTermChar;
   char                      *pUserID;
   char                      *pPassword;
   char                      *pHostName;
} db2AlertScriptAction;
```

```
typedef SQL_STRUCTURE db2AlertTaskAction
{
   char                        *pTaskName;
   db2Uint32                   condition;
   char                        *pUserID;
   char                        *pPassword;
   char                        *pHostName;
} db2AlertTaskAction;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2UpdateAlertCfgData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iObjType
Input. Specifies the type of object for which configuration is requested. Valid values are:

- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES
- DB2ALERTCFG_OBJTYPE_TABLESPACES
- DB2ALERTCFG_OBJTYPE_TS_CONTAINERS
- DB2ALERTCFG_OBJTYPE_DATABASE
- DB2ALERTCFG_OBJTYPE_TABLESPACE
- DB2ALERTCFG_OBJTYPE_TS_CONTAINER

piObjName
Input. The name of the table space or table space container when object type, *iObjType*, is set to DB2ALERTCFG_OBJTYPE_TABLESPACE or DB2ALERTCFG_OBJTYPE_TS_CONTAINER, otherwise set to NULL.

piDbName
Input. The alias name for the database for which configuration is requested when object type, *iObjType*, is DB2ALERTCFG_OBJTYPE_TS_CONTAINER, DB2ALERTCFG_OBJTYPE_TABLESPACE, and DB2ALERTCFG_OBJTYPE_DATABASE, otherwise set to NULL.

iIndicatorID
Input. The health indicator for which the configuration updates are to apply.

iNumIndAttribUpdates
Input. The number of alert attributes to be updated for the *iIndicatorID* health indicator.

piIndAttribUpdates
Input. A pointer to the *db2AlertAttrib* structure array.

iNumActionUpdates
Input. The number of alert actions to be updated for the *iIndicatorID* health indicator.

piActionUpdates
Input. A pointer to the *db2AlertActionUpdate* structure array.

iNumActionDeletes
Input. The number of alert actions to be deleted from the *iIndicatorID* health indicator.

piActionDeletes
Input. A pointer to the *db2AlertActionDelete* structure array.

iNumNewActions
Input. The number of new alert actions to be added to the *iIndicatorID* health indicator.

piNewActions
Input. A pointer to the *db2AlertActionNew* structure array.

iAttribID
Input. Specifies the alert attribute that will be updated. Valid values include:

- DB2ALERTCFG_ALARM
- DB2ALERTCFG_WARNING
- DB2ALERTCFG_SENSITIVITY
- DB2ALERTCFG_ACTIONS_ENABLED
- DB2ALERTCFG_THRESHOLD_CHECK

piAttribValue
Input. The new value of the alert attribute. Valid values include:

- DB2ALERTCFG_ALARM
- DB2ALERTCFG_WARNING
- DB2ALERTCFG_SENSITIVITY
- DB2ALERTCFG_ACTIONS_ENABLED
- DB2ALERTCFG_THRESHOLD_CHECK

iActionType
: Input. Specifies the alert action. Valid values include:
- DB2ALERTCFG_ACTIONTYPE_SCRIPT
- DB2ALERTCFG_ACTIONTYPE_TASK

piActionName
: Input. The alert action name. The name of a script action is the absolute pathname of the script. The name of a task action is a string in the form: <task-numberical-ID>.<task-numberical-suffix>.

iCondition
: The condition on which to run the action. Valid values for threshold based health indicators are:
- DB2ALERTCFG_CONDITION_ALL
- DB2ALERTCFG_CONDITION_WARNING
- DB2ALERTCFG_CONDITION_ALARM

For state based health indicators, use the numerical value defined in `sqlmon`.

iNumParmUpdates
: Input. The number of action attributes to be updated in the *piParmUpdates* array.

piParmUpdates
: Input. A pointer to the *db2AlertAttrib* structure.

piName
: Input. The name of the alert action or the script action. The name of the script action is the absolute pathname of the script, whereas the name of the task action is a string in the form: <task-numerical-ID>.<task-numerical-suffix>.

piScriptAttribs
: Input. A pointer to the *db2AlertScriptAction* structure.

piTaskAttribs
: Input. A pointer to the *db2AlertTaskAction* structure.

scriptType
: Specifies whether the script is a DB2 Command script or an operating system script. Valid values are:
- DB2ALERTCFG_SCRIPTTYPE_DB2CMD
- DB2ALERTCFG_SCRIPTTYPE_OS

condition
: The condition on which to run the action. Valid values for threshold based health indicators are:

- DB2ALERTCFG_CONDITION_ALL
- DB2ALERTCFG_CONDITION_WARNING
- DB2ALERTCFG_CONDITION_ALARM

For state based health indicators, use the numerical value defined in `sqlmon`.

pPathName
Absolute path name of the script to execute.

pWorkingDir
The absolute pathname of the directory in which the script will execute.

pCmdLineParms
The command line parameters when *scriptType* is DB2ALERTCFG_SCRIPTTYPE_OSCMD.

stmtTermChar
The character that terminates each statement in the DB2 command script when *scriptType* is DB2ALERTCFG_SCRIPTTYPE_OS.

pUserID
The user account that will execute the script.

pPassword
The valid password for *pUserId*.

pHostName
The hostName on which to run the script or task on. See db2GetAlertCfg for a description of *pHostName*.

pTaskName
The task name.

Related reference:

- "db2GetAlertCfg - Get Alert Configuration" on page 60
- "db2ResetAlertCfg - Reset Alert Configuration" on page 202

db2UpdateContact - Update Contact

Updates the attributes of a contact. Contacts are users to whom notification messages can be sent. Contacts can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

Authorization:

None

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2UpdateContact */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateContact (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateContactData
{
   char                      *piUserid;
   char                      *piPassword;
   char                      *piContactName;
   db2Uint32                 iNumAttribsUpdated;
   struct db2ContactAttrib   *piAttribs;
} db2UpdateContactData;

typedef SQL_STRUCTURE db2ContactAttrib
{
   db2Uint32                 iAttribID;
   char                      *piAttribValue;
} db2ContactAttrib;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2UpdateContactData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piContactName
Input. Specifies the name of the contact to be updated.

iNumAttribsUpdated
Input. The number attributes to be updated.

piAttribs
Input. A pointer to the *db2ContactAttrib* structure.

iAttribID
Input. Specifies the contact attribute. Valid values are:
- DB2CONTACT_ADDRESS
- DB2CONTACT_TYPE
- DB2CONTACT_MAXPAGELEN
- DB2CONTACT_DESCRIPTION

piAttribValue
Input. The new value of the contact attribute.

Related reference:
- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContact - Add Contact" on page 18
- "db2DropContact - Drop Contact" on page 57
- "db2GetContacts - Get Contacts" on page 69

db2UpdateContactGroup - Update Contact Group

Updates the attributes of a contact group. A contact group contains a list of users to whom notification messages can be sent. Contact groups can be either defined locally on the system or in a global list. The setting of the DB2 administration server (DAS) configuration parameter *contact_host* determines whether the list is local or global.

Authorization:

None.

Required connection:

None.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2UpdateContactGroup */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateContactGroup (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateContactGroupData
{
   char                     *piUserId;
   char                     *piPassword;
   char                     *piGroupName;
   db2Uint32                iNumNewContacts;
   struct db2ContactTypeData *piNewContacts;
   db2Uint32                iNumDroppedContacts;
   struct db2ContactTypeData *piDroppedContacts;
   char                     *piNewDescription;
} db2UpdateContactGroupData;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                contactType;
   char                     *pName;
} db2ContactTypeData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2ResetMonitorData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piUserid
Input. The user name.

piPassword
Input. The password for *piUserid*.

piGroupName
Input. The name of the contact group to update.

iNumNewContacts
Input. The number of new contacts to be added to the group

piNewContacts
Input. A pointer to the *db2ContactTypeData* structure.

iNumDroppedContacts
Input. The number of contacts in the group to be dropped.

piDroppedContacts
Input. A pointer to the *db2ContactTypeData* structure.

piNewDescription
Input. The new description for the group. Set this parameter to NULL if the old description should not be changed.

contactType
Specifies the type of contact. Valid values are:

- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
The contact group name, or the contact name if *contactType* is set to DB2CONTACT_SINGLE.

Related reference:

- "SQLCA" on page 478
- "Location of Contact List configuration parameter - contact_host" in the *Administration Guide: Performance*
- "db2AddContactGroup - Add Contact Group" on page 20
- "db2DropContactGroup - Drop Contact Group" on page 59
- "db2GetContactGroup - Get Contact Group" on page 66
- "db2GetContactGroups - Get Contact Groups" on page 67

db2UpdateHealthNotificationList - Update Health Notification List

Updates the contact list for notification about health alerts issued by an instance.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Instance. If there is no instance attachment, a default instance attachment is created.

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: db2UpdateHealthNotificationList */
/* ... */
SQL_API_RC SQL_API_FN
db2UpdateHealthNotificationList (
  db2Uint32 versionNumber,
  void *pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2UpdateHealthNotificationListData
{
   db2Uint32                 iNumUpdates;
   struct db2HealthNotificationListUpdate *piUpdates;
} db2UpdateHealthNotificationListData;

typedef SQL_STRUCTURE db2HealthNotificationListUpdate
{
   db2Uint32                 iUpdateType;
   struct db2ContactTypeData *piContact;
} db2HealthNotificationListUpdate;

typedef SQL_STRUCTURE db2ContactTypeData
{
   db2Uint32                 contactType;
   char                      *pName;
} db2ContactTypeData;
/* ... */
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter *pParmStruct*.

pParmStruct
Input. A pointer to the *db2UpdateHealthNotificationListData* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iNumUpdates
Input. The number of updates.

piUpdates
Input. A pointer to the *db2HealthNotificationListUpdate* structure.

iUpdateType
Input. Specifies the type of update. Valid values are:
- DB2HEALTHNOTIFICATIONLIST_ADD
- DB2HEALTHNOTIFICATIONLIST_DROP

piContact
Input. A pointer to the *db2ContactTypeData* structure.

contactType
Specifies the type of contact. Valid values are:
- DB2CONTACT_SINGLE
- DB2CONTACT_GROUP

pName
The contact group name if *contactType* is set to DB2CONTACT_GROUP, or the contact name if *ioContactType* is set to DB2CONTACT_SINGLE.

Related reference:
- "SQLCA" on page 478
- "db2GetHealthNotificationList - Get Health Notification List" on page 71

sqlabndx - Bind

Invokes the bind utility, which prepares SQL statements stored in the bind file generated by the precompiler, and creates a package that is stored in the database.

Scope:

This API can be called from any database partition server in `db2nodes.cfg`. It updates the database catalogs on the catalog partition. Its effects are visible to all database partition servers.

Authorization:

One of the following:
- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists

- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sql.h */
/* API: sqlabndx */
/* ... */
SQL_API_RC SQL_API_FN
  sqlabndx (
    _SQLOLDCHAR *pBindFileName,
    _SQLOLDCHAR *pMsgFileName,
    struct sqlopt *pBindOptions,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgbndx */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbndx (
    unsigned short MsgFileNameLen,
    unsigned short BindFileNameLen,
    struct sqlca *pSqlca,
    struct sqlopt *pBindOptions,
    _SQLOLDCHAR *pMsgFileName,
    _SQLOLDCHAR *pBindFileName);
/* ... */
```

API parameters:

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length of the message file name in bytes.

BindFileNameLen
Input. A 2-byte unsigned integer representing the length of the bind file name in bytes.

pSqlca
Output. A pointer to the *sqlca* structure.

pBindOptions
Input. A structure used to pass bind options to the API. For more information about this structure, see SQLOPT.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pBindFileName
Input. A string containing the name of the bind file, or the name of a file containing a list of bind file names. The bind file names must contain the extension `.bnd`. A path for these files can be specified.

Precede the name of a bind list file with the at sign (@). For example, a fully qualified bind list file name might be:

```
/u/user1/bnd/@all.lst
```

The bind list file should contain one or more bind file names, and must have the extension `.lst`.

Precede all but the first bind file name with a plus symbol (+). The bind file names may be on one or more lines. For example, the bind list file `all.lst` might contain:

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

Path specifications on bind file names in the list file can be used. If no path is specified, the database manager takes path information from the bind list file.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

Binding can be done as part of the precompile process for an application program source file, or as a separate step at a later time. Use BIND when binding is performed as a separate process.

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called `myapp.sqc` generates a default bind file called `myapp.bnd` and a default package name of `MYAPP`. (However, the bind file name and the package name can be overridden at precompile time by using the SQL_BIND_OPT and the SQL_PKG_OPT options of sqlaprep.)

BIND executes under the transaction that the user has started. After performing the bind, BIND issues a COMMIT (if bind is successful) or a ROLLBACK (if bind is unsuccessful) operation to terminate the current transaction and start another one.

Binding halts if a fatal error or more than 100 errors occur. If a fatal error occurs during binding, BIND stops binding, attempts to close all files, and discards the package.

Binding application programs have prerequisite requirements and restrictions beyond the scope of this manual. For example, an application cannot be bound from a V8 client to a V8 server, and then executed against a V7 server.

The Bind option types and values are defined in `sql`.

Related reference:
- "sqlaprep - Precompile Program" on page 259
- "SQLCA" on page 478
- "SQLCHAR" on page 479
- "SQLOPT" on page 521

Related samples:
- "dbpkg.sqc -- How to work with packages (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbpkg.sqC -- How to work with packages (C++)"

sqlaintp - Get Error Message

Retrieves the message associated with an error condition specified by the *sqlcode* field of the *sqlca* structure.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
/* File: sql.h */
/* API: sqlaintp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaintp (
    char *pBuffer,
    short BufferSize,
    short LineWidth,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgintp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintp (
    short BufferSize,
    short LineWidth,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pBuffer);
/* ... */
```

API parameters:

BufferSize
Input. Size, in bytes, of a string buffer to hold the retrieved message text.

LineWidth
Input. The maximum line width for each line of message text. Lines

are broken on word boundaries. A value of zero indicates that the message text is returned without line breaks.

pSqlca
Output. A pointer to the *sqlca* structure.

pBuffer
Output. A pointer to a string buffer where the message text is placed. If the message must be truncated to fit in the buffer, the truncation allows for the null string terminator character.

REXX API syntax:

```
GET MESSAGE INTO :msg [LINEWIDTH width]
```

REXX API parameters:

msg REXX variable into which the text message is placed.

width Maximum line width for each line in the text message. The line is broken on word boundaries. If *width* is not given or set to `0`, the message text returns without line breaks.

Usage notes:

One message is returned per call.

A new line (line feed, LF, or carriage return/line feed, CR/LF) sequence is placed at the end of each message.

If a positive line width is specified, new line sequences are inserted between words so that the lines do not exceed the line width.

If a word is longer than a line width, the line is filled with as many characters as will fit, a new line is inserted, and the remaining characters are placed on the next line.

In a multi-threaded application, sqlaintp must be attached to a valid context; otherwise, the message text for `SQLCODE -1445` cannot be obtained

Return codes:

Code	Message
+i	Positive integer indicating the number of bytes in the formatted message. If this is greater than the buffer size input by the caller, the message is truncated.
-1	Insufficient memory available for message formatting services to function. The requested message is not returned.
-2	No error. The *sqlca* did not contain an error code (SQLCODE = 0).
-3	Message file inaccessible or incorrect.
-4	Line width is less than zero.
-5	Invalid *sqlca*, bad buffer address, or bad buffer length.

If the return code is `-1` or `-3`, the message buffer will contain additional information about the problem.

Related reference:

- "sqlogstt - Get SQLSTATE Message" on page 399
- "SQLCA" on page 478

Related samples:

- "checkerr.cbl -- Checks for and prints to the screen SQL warnings and errors (IBM COBOL)"
- "dbcfg.sqc -- Configure database and database manager configuration parameters (C)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "dbcfg.sqC -- Configure database and database manager configuration parameters (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

sqlaprep - Precompile Program

Processes an application program source file containing embedded SQL statements. A modified source file is produced containing host language calls for the SQL statements and, by default, a package is created in the database.

Scope:

This API can be called from any database partition server in `db2nodes.cfg`. It updates the database catalogs on the catalog partition. Its effects are visible to all database partition servers.

Authorization:

One of the following:
- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sql.h */
/* API: sqlaprep */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaprep (
    _SQLOLDCHAR *pProgramName,
    _SQLOLDCHAR *pMsgFileName,
    struct sqlopt *pPrepOptions,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgprep */
/* ... */
```

```
SQL_API_RC SQL_API_FN
  sqlgprep (
    unsigned short MsgFileNameLen,
    unsigned short ProgramNameLen,
    struct sqlca *pSqlca,
    struct sqlopt *pPrepOptions,
    _SQLOLDCHAR *pMsgFileName,
    _SQLOLDCHAR *pProgramName);
/* ... */
```

API parameters:

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length of the message file name in bytes.

ProgramNameLen
Input. A 2-byte unsigned integer representing the length of the program name in bytes.

pSqlca
Output. A pointer to the *sqlca* structure.

pPrepOptions
Input. A structure used to pass precompile options to the API. For more information about this structure, see SQLOPT.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pProgramName
Input. A string containing the name of the application to be precompiled. Use the following extensions:

- `.sqb` - for COBOL applications
- `.sqc` - for C applications
- `.sqC` - for UNIX C++ applications
- `.sqf` - for FORTRAN applications
- `.sqx` - for C++ applications

When the TARGET option is used, the input file name extension does not have to be from this predefined list.

The preferred extension for C++ applications containing embedded SQL on UNIX based systems is `sqC`; however, the `sqx` convention, which was invented for systems that are not case sensitive, is tolerated by UNIX based systems.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

A modified source file is produced, which contains host language equivalents to the SQL statements. By default, a package is created in the database to which a connection has been established. The name of the package is the same as the program file name (minus the extension and folded to uppercase), up to a maximum of 8 characters.

Following connection to a database, **sqlaprep** executes under the transaction that was started. PRECOMPILE PROGRAM then issues a COMMIT or a ROLLBACK operation to terminate the current transaction and start another one.

Precompiling stops if a fatal error or more than 100 errors occur. If a fatal error does occur, PRECOMPILE PROGRAM stops precompiling, attempts to close all files, and discards the package.

The Precompile option types and values are defined in `sql.h`.

Related reference:
- "sqlabndx - Bind" on page 253
- "SQLCA" on page 478
- "SQLOPT" on page 521

Related samples:
- "dbpkg.sqc -- How to work with packages (C)"
- "dbpkg.sqC -- How to work with packages (C++)"

sqlarbnd - Rebind

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization:

One of the following:
- *sysadm* or *dbadm* authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

The authorization ID logged in the BOUNDBY column of the SYSCAT.PACKAGES system catalog table, which is the ID of the most recent binder of the package, is used as the binder authorization ID for the rebind, and for the default *schema* for table references in the package. Note that this default qualifier may be different from the authorization ID of the user executing the rebind request. REBIND will use the same bind options that were specified when the package was created.

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sql.h */
/* API: sqlarbnd */
/* ... */
SQL_API_RC SQL_API_FN
sqlarbnd (
  char *pPackageName,
  struct sqlca *pSqlca,
  struct sqlopt *pRebindOptions);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlgrbnd */
/* ... */
SQL_API_RC SQL_API_FN
sqlgrbnd (
  unsigned short PackageNameLen,
  char *pPackageName,
  struct sqlca *pSqlca,
  struct sqlopt *pRebindOptions);
/* ... */
```

API parameters:

PackageNameLen
: Input. A 2-byte unsigned integer representing the length of the package name in bytes.

pPackageName
: Input. A string containing the qualified or unqualified name that designates the package to be rebound. An unqualified package-name is implicitly qualified by the current authorization ID. This name does

not include the package version. When specifying a package that has a version that is not the empty string, then the version-id must be specified using the SQL_OPT_VERSION rebind option.

pSqlca
Output. A pointer to the *sqlca* structure.

pRebindOptions
Input. A pointer to the *SQLOPT* structure, used to pass rebind options to the API. For more information about this structure, see SQLOPT.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables ″what if″ analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. It also permits multiple rebinds within a unit of work.

This API:

- Provides a quick way to recreate a package. This enables the user to take advantage of a change in the system without a need for the original bind file. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, REBIND can be used to recreate the package. REBIND can also be used to recreate packages after db2Runstats has been executed, thereby taking advantage of the new statistics.
- Provides a method to recreate inoperative packages. Inoperative packages must be explicitly rebound by invoking either the bind utility or the rebind utility. A package will be marked inoperative (the VALID column of the `SYSCAT.PACKAGES` system catalog will be set to `X`) if a function instance on which the package depends is dropped. The rebind conservative option is not supported for inoperative packages.
- Gives users control over the rebinding of invalid packages. Invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This may result in a noticeable delay in the execution of the first SQL request for the invalid package. It may be desirable to explicitly rebind invalid packages, rather than allow the system to automatically rebind them, in order to eliminate the initial delay and to prevent unexpected SQL error messages which may be returned in case the implicit rebind fails. For example, following migration, all packages stored in the database will be invalidated by the DB2 Version 5 migration process. Given that this may involve a large number of packages, it may be

desirable to explicitly rebind all of the invalid packages at one time. This explicit rebinding can be accomplished using BIND, REBIND, or the **db2rbind** tool.

The choice of whether to use BIND or REBIND to explicitly rebind a package depends on the circumstances. It is recommended that REBIND be used whenever the situation does not specifically require the use of BIND, since the performance of REBIND is significantly better than that of BIND. BIND *must* be used, however:

- When there have been modifications to the program (for example, when SQL statements have been added or deleted, or when the package does not match the executable for the program).
- When the user wishes to modify any of the bind options as part of the rebind. REBIND does not support any bind options. For example, if the user wishes to have privileges on the package granted as part of the bind process, BIND must be used, since it has an SQL_GRANT_OPT option.
- When the package does not currently exist in the database.
- When detection of *all* bind errors is desired. REBIND only returns the first error it detects, and then ends, whereas the BIND command returns the first 100 errors that occur during binding.

REBIND is supported by DB2 Connect.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the SYSCAT.PACKAGES system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the SYSCAT.STATEMENTS system catalog table.

If many versions with the same package number and creator exist, only one version can be bound at once. If not specified using the SQL_OPT_VERSION rebind option, the VERSION defaults to be "". Even if there is only one package with a name and creator that matches the name and creator specified in the rebind request, it will not rebound unless its VERSION matches the VERSION specified explicitly or implicitly.

If REBIND encounters an error, processing stops, and an error message is returned.

The Explain tables are populated during REBIND if either SQL_EXPLSNAP_OPT or SQL_EXPLAIN_OPT have been set to YES or ALL (check EXPLAIN_SNAPSHOT and EXPLAIN_MODE columns in the catalog). The Explain tables used are those of the REBIND requester, not the original binder.

The Rebind option types and values are defined in `sql.h`.

Related tasks:
- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:
- "sqlabndx - Bind" on page 253
- "SQLCA" on page 478
- "SQLOPT" on page 521
- "REBIND Command" in the *Command Reference*
- "db2rbind - Rebind all Packages Command" in the *Command Reference*
- "db2Runstats - Runstats" on page 228

Related samples:
- "dbpkg.sqc -- How to work with packages (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbpkg.sqC -- How to work with packages (C++)"
- "rebind.sqb -- How to rebind a package (IBM COBOL)"

sqlbctcq - Close Table Space Container Query

Ends a table space container query request and frees the associated resources.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbctcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctcq (
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgctcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctcq (
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "sqlbotcq - Open Table Space Container Query" on page 277
- "sqlbftcq - Fetch Table Space Container Query" on page 269
- "sqlbtcq - Table Space Container Query" on page 287
- "sqlbstsc - Set Table Space Containers" on page 284
- "SQLCA" on page 478

Related samples:

- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbctsq - Close Table Space Query

Ends a table space query request, and frees up associated resources.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbctsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctsq (
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgctsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctsq (
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

Related reference:
- "sqlbotsq - Open Table Space Query" on page 280

- "sqlbftpq - Fetch Table Space Query" on page 271
- "sqlbmtsq - Table Space Query" on page 275
- "sqlbgtss - Get Table Space Statistics" on page 273
- "sqlbstpq - Single Table Space Query" on page 282
- "SQLCA" on page 478

Related samples:

- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbftcq - Fetch Table Space Container Query

Fetches a specified number of rows of table space container query data, each row consisting of data for a container.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

sqlbftcq - Fetch Table Space Container Query

```
/* File: sqlutil.h */
/* API: sqlbftcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftcq (
    struct sqlca *pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData,
    sqluint32 *pNumContainers);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgftcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftcq (
    struct sqlca *pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData,
    sqluint32 *pNumContainers);
/* ... */
```

API parameters:

pSqlca
: Output. A pointer to the *sqlca* structure.

MaxContainers
: Input. The maximum number of rows of data that the user allocated output area (pointed to by *pContainerData*) can hold.

pContainerData
: Output. Pointer to the output area, a structure for query data. For more information about this structure, see SQLB-TBSCONTQRY-DATA. The caller of this API must allocate space for *MaxContainers* of these structures, and set *pContainerData* to point to this space. The API will use this space to return the table space container data.

pNumContainers
: Output. Number of rows of output returned.

Usage notes:

The user is responsible for allocating and freeing the memory pointed to by the *pContainerData* parameter. This API can only be used after a successful sqlbotcq call. It can be invoked repeatedly to fetch the list generated by sqlbotcq.

Related reference:

- "sqlbotcq - Open Table Space Container Query" on page 277

- "sqlbctcq - Close Table Space Container Query" on page 266
- "sqlbtcq - Table Space Container Query" on page 287
- "sqlbstsc - Set Table Space Containers" on page 284
- "SQLCA" on page 478
- "SQLB-TBSCONTQRY-DATA" on page 471

Related samples:

- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbftpq - Fetch Table Space Query

Fetches a specified number of rows of table space query data, each row consisting of data for a table space.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbftpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftpq (
    struct sqlca *pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 *pNumTablespaces);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgftpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftpq (
    struct sqlca *pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 *pNumTablespaces);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

MaxTablespaces
Input. The maximum number of rows of data that the user allocated output area (pointed to by *pTablespaceData*) can hold.

pTablespaceData
Input and output. Pointer to the output area, a structure for query data. For more information about this structure, see SQLB-TBSPQRY-DATA. The caller of this API must:

- Allocate space for *MaxTablespaces* of these structures
- Initialize the structures
- Set TBSPQVER in the first structure to `SQLB_TBSPQRY_DATA_ID`
- Set *pTablespaceData* to point to this space. The API will use this space to return the table space data.

pNumTablespaces
Output. Number of rows of output returned.

Usage notes:

The user is responsible for allocating and freeing the memory pointed to by the *pTablespaceData* parameter. This API can only be used after a successful sqlbotsq call. It can be invoked repeatedly to fetch the list generated by sqlbotsq.

Related reference:

- "sqlbotsq - Open Table Space Query" on page 280
- "sqlbctsq - Close Table Space Query" on page 268
- "sqlbmtsq - Table Space Query" on page 275
- "sqlbgtss - Get Table Space Statistics" on page 273
- "sqlbstpq - Single Table Space Query" on page 282
- "SQLCA" on page 478
- "SQLB-TBSPQRY-DATA" on page 473

Related samples:

- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbgtss - Get Table Space Statistics

Provides information on the space utilization of a table space.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

sqlbgtss - Get Table Space Statistics

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbgtss */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbgtss (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS *pTablespaceStats);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlggtss */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtss (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS *pTablespaceStats);
/* ... */
```

API parameters:

pSqlca
: Output. A pointer to the *sqlca* structure.

TablespaceId
: Input. ID of the single table space to be queried.

pTablespaceStats
: Output. A pointer to a user-allocated *SQLB_TBS_STATS* structure. The information about the table space is returned in this structure.

Usage notes:

See SQLB-TBS-STATS for information about the fields returned and their meaning.

Related reference:

- "sqlbotsq - Open Table Space Query" on page 280
- "sqlbftpq - Fetch Table Space Query" on page 271
- "sqlbctsq - Close Table Space Query" on page 268

- "sqlbmtsq - Table Space Query" on page 275
- "sqlbstpq - Single Table Space Query" on page 282
- "SQLCA" on page 478
- "SQLB-TBS-STATS" on page 469

Related samples:
- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbmtsq - Table Space Query

Provides a one-call interface to the table space query data. The query data for all table spaces in the database is returned in an array.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbmtsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbmtsq (
    struct sqlca *pSqlca,
    sqluint32 *pNumTablespaces,
    struct SQLB_TBSPQRY_DATA ***pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgmtsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmtsq (
    struct sqlca *pSqlca,
    sqluint32 *pNumTablespaces,
    struct SQLB_TBSPQRY_DATA ***pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pNumTablespaces
Output. The total number of table spaces in the connected database.

pppTablespaceData
Output. The caller supplies the API with the address of a pointer. The space for the table space query data is allocated by the API, and a pointer to that space is returned to the caller. On return from the call, the pointer points to an array of *SQLB_TBSPQRY_DATA* pointers to the complete set of table space query data.

reserved1
Input. Always `SQLB_RESERVED1`.

reserved2
Input. Always `SQLB_RESERVED2`.

Usage notes:

This API uses the lower level services, namely:

- sqlbotsq
- sqlbftpq

- sqlbctsq

to get all of the table space query data at once.

If sufficient memory is available, this function returns the number of table spaces, and a pointer to the memory location of the table space query data. It is the user's responsibility to free this memory with a call to sqlefmem.

If sufficient memory is not available, this function simply returns the number of table spaces, and no memory is allocated. If this should happen, use sqlbotsq, sqlbftpq, and sqlbctsq, to fetch less than the whole list at once.

Related reference:

- "sqlbotsq - Open Table Space Query" on page 280
- "sqlbftpq - Fetch Table Space Query" on page 271
- "sqlbctsq - Close Table Space Query" on page 268
- "sqlbgtss - Get Table Space Statistics" on page 273
- "sqlbstpq - Single Table Space Query" on page 282
- "sqlefmem - Free Memory" on page 346
- "SQLCA" on page 478

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"

sqlbotcq - Open Table Space Container Query

Prepares for a table space container query operation, and returns the number of containers currently in the table space.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

sqlbotcq - Open Table Space Container Query

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbotcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgotcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers);
/* ... */
```

API parameters:

pSqlca
: Output. A pointer to the *sqlca* structure.

TablespaceId
: Input. ID of the table space for which container data is desired. If the special identifier `SQLB_ALL_TABLESPACES` (in `sqlutil.h`) is specified, a complete list of containers for the entire database is produced.

pNumContainers
: Output. The number of containers in the specified table space.

Usage notes:

This API is normally followed by one or more calls to sqlbftcq, and then by one call to sqlbctcq.

An application can use the following APIs to fetch information about containers in use by table spaces:

- sqlbtcq

  Fetches a complete list of container information. The API allocates the space required to hold the information for all the containers, and returns a pointer to this information. Use this API to scan the list of containers for specific information. Using this API is identical to calling the three APIs below (sqlbotcq, sqlbftcq, sqlbctcq), except that this API automatically allocates the memory for the output information. A call to this API must be followed by a call to sqlefmem to free the memory.
- sqlbotcq
- sqlbftcq
- sqlbctcq

  These three APIs function like an SQL cursor, in that they use the OPEN/FETCH/CLOSE paradigm. The caller must provide the output area for the fetch. Unlike an SQL cursor, only one table space container query can be active at a time. Use this set of APIs to scan the list of table space containers for specific information. These APIs allows the user to control the memory requirements of an application (compared with sqlbtcq).

When sqlbotcq is called, a snapshot of the current container information is formed in the agent servicing the application. If the application issues a second table space container query call (sqlbtcq or sqlbotcq), this snapshot is replaced with refreshed information.

No locking is performed, so the information in the buffer may not reflect changes made by another application after the snapshot was generated. The information is not part of a transaction.

There is one snapshot buffer for table space queries and another for table space container queries. These buffers are independent of one another.

Related reference:

- “sqlbftcq - Fetch Table Space Container Query” on page 269
- “sqlbctcq - Close Table Space Container Query” on page 266
- “sqlbtcq - Table Space Container Query” on page 287
- “sqlbstsc - Set Table Space Containers” on page 284
- “sqlefmem - Free Memory” on page 346
- “SQLCA” on page 478

Related samples:

- “tabscont.sqb -- How to get tablespace container information (IBM COBOL)”
- “tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)”

- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbotsq - Open Table Space Query

Prepares for a table space query operation, and returns the number of table spaces currently in the database.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbotsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotsq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 *pNumTablespaces);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgotsq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotsq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 *pNumTablespaces);
/* ... */
```

API parameters:

pSqlca
: Output. A pointer to the *sqlca* structure.

TablespaceQueryOptions
: Input. Indicates which table spaces to process. Valid values (defined in `sqlutil`) are:

  SQLB_OPEN_TBS_ALL
  : Process all the table spaces in the database.

  SQLB_OPEN_TBS_RESTORE
  : Process only the table spaces that the user's agent is restoring.

pNumTablespaces
: Output. The number of table spaces in the connected database.

Usage notes:

This API is normally followed by one or more calls to sqlbftpq, and then by one call to sqlbctsq.

An application can use the following APIs to fetch information about the currently defined table spaces:

- sqlbstpq

  Fetches information about a given table space. Only one table space entry is returned (into a space provided by the caller). Use this API when the table space identifier is known, and information about only that table space is desired.
- sqlbmtsq

  Fetches information about all table spaces. The API allocates the space required to hold the information for all table spaces, and returns a pointer to this information. Use this API to scan the list of table spaces when searching for specific information. Using this API is identical to calling the three APIs below, except that this API automatically allocates the memory for the output information. A call to this API must be followed by a call to sqlefmem to free the memory.
- sqlbotsq
- sqlbftpq
- sqlbctsq

  These three APIs function like an SQL cursor, in that they use the OPEN/FETCH/CLOSE paradigm. The caller must provide the output area for the fetch. Unlike an SQL cursor, only one table space query may be active at a time. Use this set of APIs to scan the list of table spaces when searching for specific information. This set of APIs allows the user to control the memory requirements of an application (compared with sqlbmtsq).

When sqlbotsq is called, a snapshot of the current table space information is buffered in the agent servicing the application. If the application issues a second table space query call (sqlbmtsq or sqlbotsq), this snapshot is replaced with refreshed information.

No locking is performed, so the information in the buffer may not reflect more recent changes made by another application. The information is not part of a transaction.

There is one snapshot buffer for table space queries and another for table space container queries. These buffers are independent of one another.

Related reference:

- "sqlbftpq - Fetch Table Space Query" on page 271
- "sqlbctsq - Close Table Space Query" on page 268
- "sqlbmtsq - Table Space Query" on page 275
- "sqlbstpq - Single Table Space Query" on page 282
- "sqlefmem - Free Memory" on page 346
- "SQLCA" on page 478

Related samples:

- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbstpq - Single Table Space Query

Retrieves information about a single currently defined table space.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbstpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstpq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32 reserved);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgstpq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstpq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA *pTablespaceData,
    sqluint32reserved);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

TablespaceId
Input. Identifier for the table space which is to be queried.

pTablespaceData
Input and output. Pointer to a user-supplied *SQLB_TBSPQRY_DATA* structure where the table space information will be placed upon

return. The caller of this API must initialize the structure and set TBSPQVER to `SQLB_TBSPQRY_DATA_ID` (in `sqlutil`).

reserved
Input. Always `SQLB_RESERVED1`.

Usage notes:

This API retrieves information about a single table space if the table space identifier to be queried is known. This API provides an alternative to the more expensive OPEN TABLESPACE QUERY, FETCH, and CLOSE combination of APIs, which must be used to scan for the desired table space when the table space identifier is not known in advance. The table space IDs can be found in the system catalogs. No agent snapshot is taken; since there is only one entry to return, it is returned directly.

For more information, see sqlbotsq.

Related reference:
- "sqlbotsq - Open Table Space Query" on page 280
- "sqlbftpq - Fetch Table Space Query" on page 271
- "sqlbctsq - Close Table Space Query" on page 268
- "sqlbmtsq - Table Space Query" on page 275
- "sqlbgtss - Get Table Space Statistics" on page 273
- "SQLCA" on page 478

Related samples:
- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbstsc - Set Table Space Containers

This API facilitates the provision of a *redirected* restore, in which the user is restoring a database, and a different set of operating system storage containers is desired or required.

Use this API when the table space is in a *storage definition pending* or a *storage definition allowed* state. These states are possible during a restore operation, immediately prior to the restoration of database pages.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbstsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstsc (
    struct sqlca *pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgstsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstsc (
    struct sqlca *pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA *pContainerData);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

SetContainerOptions
Input. Use this field to specify additional options. Valid values (defined in `sqlutil`) are:

SQLB_SET_CONT_INIT_STATE
Redo alter table space operations when performing a roll forward.

SQLB_SET_CONT_FINAL_STATE
Ignore alter table space operations in the log when performing a roll forward.

TablespaceId
Input. Identifier for the table space which is to be changed.

NumContainers
Input. The number of rows the structure pointed to by *pContainerData* holds.

pContainerData
Input. Container specifications. Although the *SQLB_TBSCONTQRY_DATA* structure is used, only the *contType*, *totalPages*, *name*, and *nameLen* (for languages other than C) fields are used; all other fields are ignored.

Usage notes:

This API is used in conjunction with db2Restore.

A backup of a database, or one or more table spaces, keeps a record of all the table space containers in use by the table spaces being backed up. During a restore, all containers listed in the backup are checked to see if they currently exist and are accessible. If one or more of the containers is inaccessible for any reason, the restore will fail. In order to allow a restore in such a case, the redirecting of table space containers is supported during the restore. This support includes adding, changing, or removing of table space containers. It is this API that allows the user to add, change or remove those containers.

Typical use of this API would involve the following sequence of actions:

1. Invoke db2Restore with *CallerAction* set to SQLUD_RESTORE_STORDEF.

   The restore utility returns an *sqlcode* indicating that some of the containers are inaccessible.
2. Invoke sqlbstsc to set the table space container definitions with the *SetContainerOptions* parameter set to SQLB_SET_CONT_FINAL_STATE.
3. Invoke sqlurst a second time with *CallerAction* set to SQLUD_CONTINUE.

The above sequence will allow the restore to use the new table space container definitions and will ignore table space add container operations in the logs when db2Rollforward is called after the restore is complete.

The user of this API should be aware that when setting the container list, there must be sufficient disk space to allow for the restore or rollforward operation to replace all of the original data into these new containers. If there is not sufficient space, such table spaces will be left in the *recovery pending* state until sufficient disk space is made available. A prudent Database Administrator will keep records of disk utilization on a regular basis. Then, when a restore or rollforward operation is needed, the required disk space will be known.

Related reference:

- "db2Rollforward - Rollforward Database" on page 219
- "SQLCA" on page 478
- "db2Backup - Backup database" on page 31
- "db2Restore - Restore database" on page 208

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"

sqlbtcq - Table Space Container Query

Provides a one-call interface to the table space container query data. The query data for all containers in a table space, or for all containers in all table spaces, is returned in an array.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

sqlbtcq - Table Space Container Query

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlbtcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbtcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers,
    struct SQLB_TBSCONTQRY_DATA **ppContainerData);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgtcq */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgtcq (
    struct sqlca *pSqlca,
    sqluint32 TablespaceId,
    sqluint32 *pNumContainers,
    struct SQLB_TBSCONTQRY_DATA **ppContainerData);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

TablespaceId
Input. ID of the table space for which container data is desired, or a special ID, `SQLB_ALL_TABLESPACES` (defined in `sqlutil`), which produces a list of all containers for the entire database.

pNumContainers
Output. The number of containers in the table space.

ppContainerData
Output. The caller supplies the API with the address of a pointer to a *SQLB_TBSCONTQRY_DATA* structure. The space for the table space container query data is allocated by the API, and a pointer to that space is returned to the caller. On return from the call, the pointer to

the *SQLB_TBSCONTQRY_DATA* structure points to the complete set of table space container query data.

Usage notes:

This API uses the lower level services, namely:
- sqlbotcq
- sqlbftcq
- sqlbctcq

to get all of the table space container query data at once.

If sufficient memory is available, this function returns the number of containers, and a pointer to the memory location of the table space container query data. It is the user's responsibility to free this memory with a call to sqlefmem.

If sufficient memory is not available, this function simply returns the number of containers, and no memory is allocated. If this should happen, use sqlbotcq, sqlbftcq, and sqlbctcq to fetch less than the whole list at once.

Related reference:
- "sqlbotcq - Open Table Space Container Query" on page 277
- "sqlbftcq - Fetch Table Space Container Query" on page 269
- "sqlbctcq - Close Table Space Container Query" on page 266
- "sqlbstsc - Set Table Space Containers" on page 284
- "sqlefmem - Free Memory" on page 346
- "SQLCA" on page 478

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"
- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"

sqlcspqy - List DRDA Indoubt Transactions

Provides a list of transactions that are indoubt between partner LUs connected by LU 6.2 protocols.

Authorization:

sysadm

Required connection:

Instance

API include file:

sqlxa.h

C API syntax:

```
/* File: sqlxa.h */
/* API: sqlcspqy */
/* ... */
extern int SQL_API_FN sqlcspqy(SQLCSPQY_INDOUBT     **indoubt_data,
                               sqlint32             *indoubt_count,
                               struct sqlca          *sqlca);
/* ... */
```

API parameters:

indoubt_data
Output. A pointer to the returned array.

indoubt_count
Output. The number of elements in the returned array.

pSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

DRDA indoubt transactions occur when communication is lost between coordinators and participants in distributed units of work.

A distributed unit of work lets a user or application read and update data at multiple locations within a single unit of work. Such work requires a two-phase commit.

The first phase requests all the participants to prepare for commit. The second phase commits or rolls back the transactions. If a coordinator or participant becomes unavailable after the first phase then the distributed transactions are indoubt.

Before issuing LIST DRDA INDOUBT TRANSACTIONS, the application process must be connected to the Sync Point Manager (SPM) instance. Use the SPM_NAME as the *dbalias* on the CONNECT statement. SPM_NAME is a database manager configuration parameter.

Related reference:

- "Sync Point Manager Name configuration parameter - spm_name" in the *Administration Guide: Performance*
- "CONNECT (Type 1) statement" in the *SQL Reference, Volume 2*
- "CONNECT (Type 2) statement" in the *SQL Reference, Volume 2*
- "SQLCA" on page 478

sqle_activate_db - Activate Database

Activates the specified database and starts up all necessary database services, so that the database is available for connection and use by any application.

Scope:

This API activates the specified database on all database partition servers. If one or more of these database partition servers encounters an error during activation of the database, a warning is returned. The database remains activated on all database partition servers on which the API has succeeded.

Note: If it is the coordinator partition or the catalog partition that encounters the error, the API returns a negative *sqlcode*, and the database will not be activated on any database partition server.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None. Applications invoking ACTIVATE DATABASE cannot have any existing database connections.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqle_activate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_activate_db (
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlg_activate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_activate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

DbAliasLen
: Input. A 2-byte unsigned integer representing the length of the database alias name in bytes.

UserNameLen
: Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

PasswordLen
: Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

pDbAlias
Input. Pointer to the database alias name.

pUserName
Input. Pointer to the user ID starting the database. Can be NULL.

pPassword
Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

If a database has not been started, and a DB2 CONNECT TO (or an implicit connect) is encountered in an application, the application must wait while the database manager starts up the required database. In such cases, this first application spends time on database initialization before it can do any work. However, once the first application has started a database, other applications can simply connect and use it.

Database administrators can use ACTIVATE DATABASE to start up selected databases. This eliminates any application time spent on database initialization.

Databases initialized by ACTIVATE DATABASE can only be shut down by sqle_deactivate_db, or by db2InstanceStop. To obtain a list of activated databases, call db2GetSnapshot.

If a database was started by a DB2 CONNECT TO (or an implicit connect) and subsequently an ACTIVATE DATABASE is issued for that same database, then DEACTIVATE DATABASE must be used to shut down that database.

ACTIVATE DATABASE behaves in a similar manner to a DB2 CONNECT TO (or an implicit connect) when working with a database requiring a restart (for example, database in an inconsistent state). The database will be restarted before it can be initialized by ACTIVATE DATABASE.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "db2GetSnapshot - Get Snapshot" on page 73
- "sqle_deactivate_db - Deactivate Database" on page 294
- "SQLCA" on page 478
- "ACTIVATE DATABASE Command" in the *Command Reference*

sqle_deactivate_db - Deactivate Database

Stops the specified database.

Scope:

In a partitioned database environment, this API deactivates the specified database on all database partition servers. If one or more of these database partition servers encounters an error, a warning is returned. The database will be successfully deactivated on some database partition servers, but may remain activated on the database partition servers encountering the error.

Note: If it is the coordinator partition or the catalog partition that encounters the error, the API returns a negative *sqlcode*, and the database will not be reactivated on any database partition server on which it was deactivated.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None. Applications invoking DEACTIVATE DATABASE cannot have any existing database connections.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqle_deactivate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlg_deactivate_db */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_deactivate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char *pDbAlias,
    char *pUserName,
    char *pPassword,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

DbAliasLen
Input. A 2-byte unsigned integer representing the length of the database alias name in bytes.

UserNameLen
Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

PasswordLen
Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

pDbAlias
Input. Pointer to the database alias name.

pUserName
Input. Pointer to the user ID stopping the database. Can be NULL.

pPassword
Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

Databases initialized by ACTIVATE DATABASE can only be shut down by DEACTIVATE DATABASE. db2InstanceStop automatically stops all activated databases before stopping the database manager. If a database was initialized by ACTIVATE DATABASE, the last DB2 CONNECT RESET statement (counter equal 0) will not shut down the database; DEACTIVATE DATABASE must be used.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqle_activate_db - Activate Database" on page 291
- "SQLCA" on page 478
- "DEACTIVATE DATABASE Command" in the *Command Reference*

sqleaddn - Add Node

Adds a new database partition server to the partitioned database environment. This API creates database partitions for all databases currently defined in the instance on the new database partition server. The user can specify the source database partition server for any system temporary table spaces to be created with the databases, or specify that no system temporary table spaces are to be created. The API must be issued from the database partition server that is being added, and can only be issued on a database partition server.

Scope:

This API only affects the database partition server on which it is executed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleaddn */
/* ... */
SQL_API_RC SQL_API_FN
  sqleaddn (
    void *pAddNodeOptions,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgaddn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddn (
    unsigned short addnOptionsLen,
    struct sqlca *pSqlca,
    void *pAddNodeOptions);
/* ... */
```

API parameters:

addnOptionsLen
Input. A 2-byte unsigned integer representing the length of the optional *sqle_addn_options* structure in bytes.

pSqlca
Output. A pointer to the *sqlca* structure.

pAddNodeOptions
Input. A pointer to the optional *sqle_addn_options* structure. This structure is used to specify the source database partition server, if any, of the system temporary table space definitions for all database partitions created during the add node operation. If not specified (that is, a NULL pointer is specified), the system temporary table space definitions will be the same as those for the catalog partition.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

Before adding a new database partition server, ensure that there is sufficient storage for the containers that must be created for all existing databases on the system.

The add node operation creates an empty database partition on the new database partition server for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default value.

If an add node operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added (that is, the local database partition server). Existing database partitions remain unaffected on all other database partition servers. If this fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition server cannot be used to contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition server to a database partition group.

This API will fail if a create database or a drop database operation is in progress. The API can be called again once the operation has completed.

If system temporary table spaces are to be created with the database partitions, sqleaddn may have to communicate with another database partition server in the partitioned database environment in order to retrieve the table space definitions. The *start_stop_time* database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the table space definitions. If this time is exceeded, the API fails. Increase the value of *start_stop_time*, and call the API again.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "ALTER DATABASE PARTITION GROUP statement" in the *SQL Reference, Volume 2*
- "sqlecrea - Create Database" on page 314
- "sqledrpn - Drop Node Verify" on page 343
- "SQLCA" on page 478
- "SQLE-ADDN-OPTIONS" on page 485

sqleatcp - Attach and Change Password

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the **DB2INSTANCE** environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Note: This API extends the function of the sqleatin API by permitting the optional change of the user password for the instance being attached.

Authorization:

None

Required connection:

This API establishes an instance attachment.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleatcp */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatcp (
    char *pNodeName,
    char *pUserName,
    char *pPassword,
    char *pNewPassword,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgatcp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatcp (
    unsigned short NewPasswordLen,
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    char *pNewPassword,
    char *pPassword,
    char *pUserName,
    char *pNodeName);
/* ... */
```

API parameters:

NewPasswordLen
Input. A 2-byte unsigned integer representing the length of the new password in bytes. Set to zero if no new password is supplied.

PasswordLen
Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

UserNameLen
Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

NodeNameLen
Input. A 2-byte unsigned integer representing the length of the node name in bytes. Set to zero if no node name is supplied.

pSqlca
Output. A pointer to the *sqlca* structure.

pNewPassword
Input. A string containing the new password for the specified user name. Set to NULL if a password change is not required.

pPassword
Input. A string containing the password for the specified user name. May be NULL.

pUserName
Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

pNodeName
Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified

by the **DB2INSTANCE** environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory. May be NULL.

REXX API syntax:

Calling this API directly from REXX is not supported. However, REXX programmers can utilize this function by calling the DB2 command line processor to execute the ATTACH command.

Usage notes:

Note: A node name in the node directory can be regarded as an alias for an instance.

If an attach request succeeds, the *sqlerrmc* field of the *sqlca* will contain 9 tokens separated by hexadecimal FF (similar to the tokens returned when a CONNECT request is successful):

1. Country/region code of the application server
2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the API)
5. Identity and platform type of the server
6. Agent ID of the agent which has been started at the server
7. Agent index
8. Node number of the server
9. Number of partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the *sqlerrmc* field of the *sqlca* (as outlined above).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

Certain functions (**db2start**, **db2stop**, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the **DB2INSTANCE** environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated, and where the password is changed, depend on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

Related reference:

- "sqlesetc - Set Client" on page 386
- "sqleatin - Attach" on page 302
- "sqledtin - Detach" on page 345
- "SQLCA" on page 478
- "SQLE-CONN-SETTING" on page 489

Related samples:

- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"

sqleatin - Attach

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the **DB2INSTANCE** environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Note: If a password change is required, use the sqleatcp API instead of the sqleatin API.

Authorization:

None

Required connection:

This API establishes an instance attachment.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleatin */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatin (
    char *pNodeName,
    char *pUserName,
    char *pPassword,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgatin */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatin (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    char *pPassword,
    char *pUserName,
    char *pNodeName);
/* ... */
```

API parameters:

PasswordLen
: Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

UserNameLen
: Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

NodeNameLen
: Input. A 2-byte unsigned integer representing the length of the node name in bytes. Set to zero if no node name is supplied.

pSqlca
: Output. A pointer to the *sqlca* structure.

pPassword
: Input. A string containing the password for the specified user name. May be NULL.

pUserName
: Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

pNodeName
Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the **DB2INSTANCE** environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory. May be NULL.

REXX API syntax:

```
ATTACH [TO nodename [USER username USING password]]
```

REXX API parameters:

nodename
Alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the **DB2INSTANCE** environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory.

username
Name under which the user attaches to the instance.

password
Password used to authenticate the user name.

Usage notes:

Note: A node name in the node directory can be regarded as an alias for an instance.

If an attach request succeeds, the *sqlerrmc* field of the *sqlca* will contain 9 tokens separated by hexadecimal FF (similar to the tokens returned when a CONNECT request is successful):

1. Country/region code of the application server
2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the API)
5. Identity and platform type of the server
6. Agent ID of the agent which has been started at the server
7. Agent index
8. Node number of the server
9. Number of partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the *sqlerrmc* field of the *sqlca* (as outlined above).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

Certain functions (**db2start**, **db2stop**, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the **DB2INSTANCE** environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated depends on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

Related reference:
- "sqlesetc - Set Client" on page 386
- "sqledtin - Detach" on page 345
- "sqleatcp - Attach and Change Password" on page 299
- "SQLCA" on page 478
- "SQLE-CONN-SETTING" on page 489

Related samples:
- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

sqlecadb - Catalog Database

Stores database location information in the system database directory. The database can be located either on the local workstation or on a remote node.

Scope:

sqlecadb - Catalog Database

This API affects the system database directory. In a partitioned database environment, when cataloging a local database into the system database directory, this API must be called from a database partition server where the database resides.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlecadb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecadb (
    _SQLOLDCHAR *pDbName,
    _SQLOLDCHAR *pDbAlias,
    unsigned char Type,
    _SQLOLDCHAR *pNodeName,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pComment,
    unsigned short Authentication,
    _SQLOLDCHAR *pPrincipal,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgcadb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcadb (
    unsigned short PrinLen,
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short NodeNameLen,
    unsigned short DbAliasLen,
    unsigned short DbNameLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pPrinName,
    unsigned short Authentication,
    _SQLOLDCHAR *pComment,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pNodeName,
    unsigned char Type,
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pDbName);
/* ... */
```

API parameters:

PrinLen
: Input. A 2-byte unsigned integer representing the length in bytes of the principal name. Set to zero if no principal is provided. This value should be nonzero only when authentication is specified as `SQL_AUTHENTICATION_DCE` or `SQL_AUTHENTICATION_KERBEROS`.

CommentLen
: Input. A 2-byte unsigned integer representing the length in bytes of the comment. Set to zero if no comment is provided.

PathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the path of the local database directory. Set to zero if no path is provided.

NodeNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the node name. Set to zero if no node name is provided.

DbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

DbNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database name.

pSqlca
: Output. A pointer to the *sqlca* structure.

pPrinName
: Input. A string containing the principal name of the DB2 server on which the database resides. This value should only be specified when authentication is `SQL_AUTHENTICATION_DCE` or `SQL_AUTHENTICATION_KERBEROS`. For DCE, the principal must be the same as the value stored in the server's keytab file.

Authentication
: Input. Contains the authentication type specified for the database. Authentication is a process that verifies that the user is who he/she

claims to be. Access to database objects depends on the user's authentication. Valid values (from `sqlenv`) are:

SQL_AUTHENTICATION_SERVER
Specifies that authentication takes place on the node containing the target database.

SQL_AUTHENTICATION_CLIENT
Specifies that authentication takes place on the node where the application is invoked.

SQL_AUTHENTICATION_DCE
Specifies that authentication takes place using DCE Security Services.

SQL_AUTHENTICATION_KERBEROS
Specifies that authentication takes place using Kerberos Security Mechanism.

SQL_AUTHENTICATION_NOT_SPECIFIED
Authentication not specified.

SQL_AUTHENTICATION_SVR_ENCRYPT
Specifies that authentication takes place on the node containing the target database, and that the authentication password is to be encrypted.

This parameter can be set to `SQL_AUTHENTICATION_NOT_SPECIFIED`, except when cataloging a database that resides on a DB2 Version 1 server.

Specifying the authentication type in the database catalog results in a performance improvement during a connect.

pComment
Input. A string containing an optional description of the database. A null string indicates no comment. The maximum length of a comment string is 30 characters.

pPath Input. A string which, on UNIX based systems, specifies the name of the path on which the database being cataloged resides. Maximum length is 215 characters.

On the Windows operating system, this string specifies the letter of the drive on which the database being cataloged resides.

If a NULL pointer is provided, the default database path is assumed to be that specified by the database manager configuration parameter *dftdbpath*.

pNodeName
Input. A string containing the name of the node where the database is located. May be NULL.

Note: If neither *pPath* nor *pNodeName* is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter *dftdbpath*.

Type Input. A single character that designates whether the database is indirect, remote, or is cataloged via DCE. Valid values (defined in `sqlenv`) are:

SQL_INDIRECT
Specifies that the database resides at this instance.

SQL_REMOTE
Specifies that the database resides at another instance.

SQL_DCE
Specifies that the database is cataloged via DCE.

pDbAlias
Input. A string containing an alias for the database.

pDbName
Input. A string containing the database name.

REXX API syntax:

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]
```

REXX API parameters:

dbname
Name of the database to be cataloged.

alias Alternate name for the database. If an alias is not specified, the database name is used as the alias.

path Path on which the database being cataloged resides.

nodename
Name of the remote workstation where the database being cataloged resides.

Note: If neither *path* nor *nodename* is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter *dftdbpath*.

authentication
Place where authentication is to be done. Valid values are:

SERVER
Authentication occurs at the node containing the target database. This is the default.

CLIENT
Authentication occurs at the node where the application is invoked.

DCS
Specifies how authentication will take place for databases accessed using DB2 Connect. The behavior is the same as for the type `SERVER`, except that when the authentication type is `SERVER`, DB2 Connect forces authentication at the gateway, and when the authentication type is `DCS`, authentication is assumed to take place at the host.

DCE SERVER PRINCIPAL *dce_principal_name*
Fully qualified DCE principal name for the target server. This value is also recorded in the keytab file at the target server.

comment
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

REXX API syntax:

```
CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [WITH comment]
```

REXX API parameters:

db_global_name
The fully qualified name that uniquely identifies the database in the DCE name space.

alias
Alternate name for the database.

DCE
The global directory service being used.

comment
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

Examples:

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```

Usage notes:

Use CATALOG DATABASE to catalog databases located on local or remote nodes, recatalog databases that were uncataloged previously, or maintain multiple aliases for one database (regardless of database location).

DB2 automatically catalogs databases when they are created. It catalogs an entry for the database in the local database directory, and another entry in the system database directory. If the database is created from a remote client (or a client which is executing from a different instance on the same machine), an entry is also made in the system database directory at the client instance.

Databases created at the current instance (as defined by the value of the **DB2INSTANCE** environment variable) are cataloged as *indirect*. Databases created at other instances are cataloged as *remote* (even if they physically reside on the same machine).

CATALOG DATABASE automatically creates a system database directory if one does not exist. The system database directory is stored on the path that contains the database manager instance that is being used. The system database directory is maintained outside of the database. Each entry in the directory contains:

- Alias
- Authentication type
- Comment
- Database
- Entry type
- Local database directory (when cataloging a local database)
- Node name (when cataloging a remote database)
- Release information.

If a database is cataloged with the type parameter set to `SQL_INDIRECT`, the value of the authentication parameter provided will be ignored, and the authentication in the directory will be set to `SQL_AUTHENTICATION_NOT_SPECIFIED`.

If directory caching is enabled, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh

DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related reference:

- "sqledcls - Close Database Directory Scan" on page 330
- "sqledgne - Get Next Database Directory Entry" on page 331
- "sqledosd - Open Database Directory Scan" on page 334
- "sqleuncd - Uncatalog Database" on page 392
- "SQLCA" on page 478

Related samples:

- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlecran - Create Database at Node

Creates a database only on the database partition server that calls the API. This API is not intended for general use. For example, it should be used with db2Restore if the database partition at a database partition server was damaged and must be recreated. Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

Note: If this API is used to recreate a database partition that was dropped (because it was damaged), the database at this database partition server will be in the restore-pending state. After recreating the database partition, the database must immediately be restored on this database partition server.

Scope:

This API only affects the database partition server on which it is called.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

Instance. To create a database at another database partition server, it is necessary to first attach to that database partition server. A database connection is temporarily established by this API during processing.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlecran */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecran (
    char *pDbName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgcran */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcran (
    unsigned short reservedLen,
    unsigned short dbNameLen,
    struct sqlca *pSqlca,
    void *pReserved,
    char *pDbName);
/* ... */
```

API parameters:

reservedLen
: Input. Reserved for the length of *pReserved*.

dbNameLen
: Input. A 2-byte unsigned integer representing the length of the database name in bytes.

pSqlca
: Output. A pointer to the *sqlca* structure.

pReserved
: Input. A spare pointer that is set to null or points to zero. Reserved for future use.

pDbName
: Input. A string containing the name of the database to be created. Must not be NULL.

sqlecran - Create Database at Node

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

When the database is successfully created, it is placed in restore-pending state. The database must be restored on this database partition server before it can be used.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqlecrea - Create Database" on page 314
- "sqledpan - Drop Database at Node" on page 336
- "SQLCA" on page 478
- "db2Restore - Restore database" on page 208

sqlecrea - Create Database

Initializes a new database with an optional user-defined collating sequence, creates the three initial table spaces, creates the system tables, and allocates the recovery log.

Scope:

In a partitioned database environment, this API affects all database partition servers that are listed in the `db2nodes.cfg` file.

The database partition server from which this API is called becomes the catalog partition for the new database.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

Instance. To create a database at another (remote) node, it is necessary to first attach to that node. A database connection is temporarily established by this API during processing.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlecrea */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecrea (
    char *pDbName,
    char *pLocalDbAlias,
    char *pPath,
    struct sqledbdesc *pDbDescriptor,
    struct sqledbterritoryinfo *pTerritoryInfo,
    char Reserved2,
    void *pReserved1,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgcrea */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcrea (
    unsigned short PathLen,
    unsigned short LocalDbAliasLen,
    unsigned short DbNameLen,
    struct sqlca *pSqlca,
    void *pReserved1,
    unsigned short Reserved2,
    struct sqledbterritoryinfo *pTerritoryInfo,
    struct sqledbdesc *pDbDescriptor,
    char *pPath,
    char *pLocalDbAlias,
    char *pDbName);
/* ... */
```

API parameters:

PathLen
Input. A 2-byte unsigned integer representing the length of the path in bytes. Set to zero if no path is provided.

LocalDbALiasLen
: Input. A 2-byte unsigned integer representing the length of the local database alias in bytes. Set to zero if no local alias is provided.

DbNameLen
: Input. A 2-byte unsigned integer representing the length of the database name in bytes.

pSqlca
: Output. A pointer to the *sqlca* structure.

pReserved1
: Input. A spare pointer that is set to null or points to zero.

Reserved2
: Input. Reserved for future use.

pTerrirotyInfo
: Input. A pointer to the *sqledbterritoryinfo* structure, containing the locale and the code set for the database. May be NULL.

pDbDescriptor
: Input. A pointer to the database description block used when creating the database. The database description block may be used to supply values that are permanently stored in the configuration file of the database, such as collating sequence. May be NULL.

pPath Input. On UNIX based systems, specifies the path on which to create the database. If a path is not specified, the database is created on the default database path specified in the database manager configuration file (*dftdbpath* parameter). On the Windows operating system, specifies the letter of the drive on which to create the database. May be NULL.

Note: For partitioned database environments, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the *dftdbpath* database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the `$HOME` directory of the instance owner). The path specified for this API in a partitioned database environment cannot be a relative path.

pLocalDbAlias
: Input. A string containing the alias to be placed in the client's system database directory. May be NULL. If no local alias is specified, the database name is the default.

pDbName
: Input. A string containing the database name. This is the database name that will be cataloged in the system database directory. Once the

database has been successfully created in the server's system database directory, it is automatically cataloged in the system database directory with a database alias identical to the database name. Must not be NULL.

REXX API syntax:

```
CREATE DATABASE dbname [ON path] [ALIAS dbalias]
[USING CODESET codeset TERRITORY territory]
[COLLATE USING {SYSTEM | IDENTITY | USER :udcs}]
[NUMSEGS numsegs] [DFT_EXTENT_SZ dft_extentsize]
[CATALOG TABLESPACE <tablespace_definition>]
[USER TABLESPACE <tablespace_definition>]
[TEMPORARY TABLESPACE <tablespace_definition>]
[WITH comment]

Where <tablespace_definition> stands for:
MANAGED BY {
SYSTEM USING :SMS_string |
DATABASE USING :DMS_string }
[ EXTENTSIZE number_of_pages ]
[ PREFETCHSIZE number_of_pages ]
[ OVERHEAD number_of_milliseconds ]
[ TRANSFERRATE number_of_milliseconds ]
```

REXX API parameters:

dbname
Name of the database.

dbalias
Alias of the database.

path Path on which to create the database.

If a path is not specified, the database is created on the default database path specified in the database manager configuration file (*dftdbpath* configuration parameter).

Note: For partitioned database environments, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the *dftdbpath* database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the `$HOME` directory of the instance owner). The path specified for this API in a partitioned database environment cannot be a relative path.

codeset
Code set to be used for data entered into the database.

territory
Territory code (locale) to be used for data entered into the database.

SYSTEM
Uses the collating sequence of the operating system based on the current territory code.

IDENTITY
The collating sequence is the identity sequence, where strings are compared byte for byte, starting with the leftmost byte.

USER udcs
The collating sequence is specified by the calling application in a host variable containing a 256-byte string defining the collating sequence.

numsegs
Number of segment directories that will be created and used to store the DAT, IDX, and LF files.

dft_extentsize
Specifies the default *extent size* for table spaces in the database.

SMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, and where the table space data will be stored. In the following, XXX represents the host variable name. Note that each of the directory names cannot exceed 254 bytes in length.

XXX.0 Number of directories specified

XXX.1 First directory name for SMS table space

XXX.2 Second directory name for SMS table space

XXX.3 and so on.

DMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, where the table space data will be stored, container sizes (specified in a number of 4KB pages) and types (file or device). The specified devices (not files) must already exist. In the following, XXX represents the host variable name. Note that each of the container names cannot exceed 254 bytes in length.

XXX.0 Number of strings in the REXX host variable (number of first level elements)

XXX.1.1
Type of the first container (`file` or `device`)

XXX.1.2
First file name or device name

XXX.1.3
: Size (in pages) of the first container

XXX.2.1
: Type of the second container (`file` or `device`)

XXX.2.2
: Second file name or device name

XXX.2.3
: Size (in pages) of the second container

XXX.3.1
: and so on.

EXTENTSIZE number_of_pages
: Number of 4KB pages that will be written to a container before skipping to the next container.

PREFETCHSIZE number_of_pages
: Number of 4KB pages that will be read from the table space when data prefetching is being performed.

OVERHEAD number_of_milliseconds
: Number that specifies the I/O controller overhead, disk seek, and latency time in milliseconds.

TRANSFERRATE number_of_milliseconds
: Number that specifies the time in milliseconds to read one 4KB page into memory.

comment
: Description of the database or the database entry in the system directory. Do not use a carriage return or line feed character in the comment. Be sure to enclose the comment text in double quotation marks. Maximum size is 30 characters.

Usage notes:

CREATE DATABASE:

- Creates a database in the specified subdirectory. In a partitioned database environment, creates the database on all database partition servers listed in `db2nodes.cfg`, and creates a `$DB2INSTANCE/NODE`*xxxx* directory under the specified subdirectory at each database partition server, where *xxxx* represents the local database partition server number. In a single-partition environment, creates a `$DB2INSTANCE/NODE0000` directory under the specified subdirectory.
- Creates the system catalog tables and recovery log.
- Catalogs the database in the following database directories:

  - server's local database directory on the path indicated by *pPath* or, if the path is not specified, the default database path defined in the database manager system configuration file. A local database directory resides on each file system that contains a database.
  - server's system database directory for the attached instance. The resulting directory entry will contain the database name and a database alias.

    If the API was called from a remote client, the client's system database directory is also updated with the database name and an alias.

  Creates a system or a local database directory if neither exists. If specified, the comment and code set values are placed in both directories.
- Stores the specified code set, territory, and collating sequence. A flag is set in the database configuration file if the collating sequence consists of unique weights, or if it is the identity sequence.
- Creates the schemata called SYSCAT, SYSFUN, SYSIBM, and SYSSTAT with SYSIBM as the owner. The database partition server on which this API is called becomes the catalog partition for the new database. Two database partition groups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP.
- Binds the previously defined database manager bind files to the database (these are listed in `db2ubind.lst`). If one or more of these files do not bind successfully, sqlecrea returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC.
- Creates SYSCATSPACE, TEMPSPACE1, and USERSPACE1 table spaces. The SYSCATSPACE table space is only created on the catalog partition. All database partitions have the same table space definitions.
- Grants the following:
  - DBADM authority, and CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA, and LOAD privileges to the database creator
  - CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA privileges to PUBLIC
  - USE privilege on the USERSPACE1 table space to PUBLIC
  - SELECT privilege on each system catalog to PUBLIC
  - BIND and EXECUTE privilege to PUBLIC for each successfully bound utility
  - EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema.

– EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema.

With *dbadm* authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with *sysadm* or *dbadm* authority over the database revokes these privileges, the database creator nevertheless retains them.

In a partitioned database environment, the database manager creates a subdirectory, `$DB2INSTANCE/NODE`*xxxx*, under the specified or default path on all database partition servers. The *xxxx* is the node number as defined in the `db2nodes.cfg` file (that is, node 0 becomes `NODE0000`). Subdirectories `SQL00001` through `SQL`*nnnnn* will reside on this path. This ensures that the database objects associated with different database partition servers are stored in different directories (even if the subdirectory `$DB2INSTANCE` under the specified or default path is shared by all database partition servers).

CREATE DATABASE will fail if the application is already connected to a database.

If the database description block structure is not set correctly, an error message is returned.

The "eye-catcher" of the database description block must be set to the symbolic value `SQLE_DBDESC_2` (defined in `sqlenv`). The following sample user-defined collating sequences are available in the host language include files:

sqle819a	If the code page of the database is 819 (ISO Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).
sqle819b	If the code page of the database is 819 (ISO Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).
sqle850a	If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).
sqle850b	If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).
sqle932a	If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5035 (EBCDIC Japanese).
sqle932b	If the code page of the database is 932 (ASCII Japanese), this

sequence will cause sorting to be performed according to the host CCSID 5026 (EBCDIC Japanese).

The collating sequence specified during CREATE DATABASE cannot be changed later, and all character comparisons in the database use the specified collating sequence. This affects the structure of indexes as well as the results of queries.

Use **sqlecadb** to define different alias names for the new database.

Related reference:

- "sqlabndx - Bind" on page 253
- "sqlecadb - Catalog Database" on page 305
- "sqledrpd - Drop Database" on page 340
- "sqlecran - Create Database at Node" on page 312
- "sqledpan - Drop Database at Node" on page 336
- "SQLEDBTERRITORYINFO" on page 502
- "SQLCA" on page 478
- "SQLEDBDESC" on page 503

Related samples:

- "db_udcs.cbl -- How to use user-defined collating sequence (IBM COBOL)"
- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "ebcdicdb.cbl -- Create a database with EBCDIC 037 standard collating sequence (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbcreate.C -- Create and drop databases (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

sqlectnd - Catalog Node

Stores information in the node directory about the location of a DB2 server instance based on the communications protocol used to access that instance. The information is needed to establish a database connection or attachment between an application and a server instance.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlectnd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlectnd (
    struct sqle_node_struct *pNodeInfo,
    void *pProtocolInfo,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgctnd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctnd (
    struct sqlca *pSqlca,
    struct sqle_node_struct *pNodeInfo,
    void *pProtocolInfo);
/* ... */
```

API parameters:

pNodeInfo
Input. A pointer to a node directory structure.

pProtocolInfo
Input. A pointer to the protocol structure:

- SQLE-NODE-CPIC
- SQLE-NODE-IPXSPX
- SQLE-NODE-LOCAL
- SQLE-NODE-NETB
- SQLE-NODE-NPIPE
- SQLE-NODE-TCPIP.

pSqlca
: Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
CATALOG APPC NODE nodename DESTINATION symbolic_destination_name
[SECURITY {NONE|SAME|PROGRAM}]
[WITH comment]
```

REXX API parameters:

nodename
: Alias for the node to be cataloged.

symbolic_destination_name
: Symbolic destination name of the remote partner node.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

REXX API syntax:

```
CATALOG IPXSPX NODE nodename REMOTE file_server SERVER objectname
[WITH comment]
```

REXX API parameters:

nodename
: Alias for the node to be cataloged.

file_server
: Name of the NetWare file server where the internetwork address of the database manager instance is registered. The internetwork address is stored in the bindery at the NetWare file server, and is accessed using *objectname*.

objectname
: The database manager server instance is represented as the object, *objectname*, on the NetWare file server. The server's IPX/SPX internetwork address is stored and retrieved from this object.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

REXX API syntax:

```
CATALOG LOCAL NODE nodename INSTANCE instance_name [WITH comment]
```

REXX API parameters:

nodename
Alias for the node to be cataloged.

instance_name
Name of the instance to be cataloged.

comment
An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

REXX API syntax:

```
CATALOG NETBIOS NODE nodename REMOTE server_nname ADAPTER adapternum
[WITH comment]
```

REXX API parameters:

nodename
Alias for the node to be cataloged.

server_nname
Name of the remote workstation. This is the workstation name (*nname*) found in the database manager configuration file of the server instance.

adapternum
Local LAN adapter number.

comment
An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

REXX API syntax:

```
CATALOG NPIPE NODE nodename REMOTE computer_name INSTANCE instance_name
```

REXX API parameters:

nodename
Alias for the node to be cataloged.

computer_name
The computer name of the node on which the target database resides.

instance_name
: Name of the instance to be cataloged.

REXX API syntax:

```
CATALOG TCPIP NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

REXX API parameters:

nodename
: Alias for the node to be cataloged.

hostname
: Host name of the node where the target database resides.

servicename
: Either the service name of the database manager instance on the remote node, or the port number associated with that service name.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

Usage notes:

DB2 creates the node directory on the first call to this API if the node directory does not exist. On the Windows operating system, the node directory is stored in the directory of the instance being used. On UNIX based systems, it is stored in the DB2 install directory (`sqllib`, for example).

If directory caching is enabled, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related reference:

- "sqlencls - Close Node Directory Scan" on page 370
- "sqlengne - Get Next Node Directory Entry" on page 371
- "sqlenops - Open Node Directory Scan" on page 374
- "sqleuncn - Uncatalog Node" on page 394
- "SQLE-NODE-CPIC" on page 495

- "SQLE-NODE-NETB" on page 497
- "SQLE-NODE-STRUCT" on page 499
- "SQLE-NODE-TCPIP" on page 500
- "SQLCA" on page 478
- "SQLE-NODE-IPXSPX" on page 496
- "SQLE-NODE-LOCAL" on page 497
- "SQLE-NODE-NPIPE" on page 498

Related samples:

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"
- "nodecat.cbl -- Get node directory information (IBM COBOL)"

sqledcgd - Change Database Comment

Changes a database comment in the system database directory or the local database directory. New comment text can be substituted for text currently associated with a comment.

Scope:

This API only affects the database partition server on which it is issued.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledcgd */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcgd (
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pComment,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdcgd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcgd (
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pComment,
    _SQLOLDCHAR *pPath,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API parameters:

CommentLen
Input. A 2-byte unsigned integer representing the length in bytes of the comment. Set to zero if no comment is provided.

PathLen
Input. A 2-byte unsigned integer representing the length in bytes of the path parameter. Set to zero if no path is provided.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pSqlca
Output. A pointer to the *sqlca* structure.

pComment
Input. A string containing an optional description of the database. A null string indicates no comment. It can also indicate no change to an existing database comment.

pPath Input. A string containing the path on which the local database directory resides. If the specified path is a null pointer, the system database directory is used.

The comment is only changed in the local database directory or the system database directory on the database partition server on which the API is executed. To change the database comment on all database partition servers, run the API on every database partition server.

pDbAlias
Input. A string containing the database alias. This is the name that is cataloged in the system database directory, or the name cataloged in the local database directory if the path is specified.

REXX API syntax:

```
CHANGE DATABASE database_alias COMMENT [ON path] WITH comment
```

REXX API parameters:

database_alias
Alias of the database whose comment is to be changed.

To change the comment in the system database directory, it is necessary to specify the database alias.

If the path where the database resides is specified (with the *path* parameter), enter the name (not the alias) of the database. Use this method to change the comment in the local database directory.

path Path on which the database resides.

comment
Describes the entry in the system database directory or the local database directory. Any comment that helps to describe the cataloged database can be entered. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

Usage notes:

New comment text replaces existing text. To append information, enter the old comment text, followed by the new text.

Only the comment for an entry associated with the database alias is modified. Other entries with the same database name, but with different aliases, are not affected.

If the path is specified, the database alias must be cataloged in the local database directory. If the path is not specified, the database alias must be cataloged in the system database directory.

Related reference:
- "sqlecadb - Catalog Database" on page 305
- "sqledcls - Close Database Directory Scan" on page 330
- "sqlecrea - Create Database" on page 314

- "sqledgne - Get Next Database Directory Entry" on page 331
- "sqledosd - Open Database Directory Scan" on page 334

Related samples:

- "dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqledcls - Close Database Directory Scan

Frees the resources allocated by sqledosd - Open Database Directory Scan.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledcls */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcls (
    unsigned short Handle,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdcls */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcls (
    unsigned short Handle,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

Handle
Input. Identifier returned from the associated OPEN DATABASE DIRECTORY SCAN API.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
CLOSE DATABASE DIRECTORY scanid
```

REXX API parameters:

scanid A host variable containing the *scanid* returned from the OPEN DATABASE DIRECTORY SCAN API.

Related reference:

- "sqledgne - Get Next Database Directory Entry" on page 331
- "sqledosd - Open Database Directory Scan" on page 334
- "SQLCA" on page 478

Related samples:

- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"
- "dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqledgne - Get Next Database Directory Entry

Returns the next entry in the system database directory or the local database directory copy returned by sqledosd - Open Database Directory Scan. Subsequent calls to this API return additional entries.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledgne */
/* ... */
SQL_API_RC SQL_API_FN
  sqledgne (
    unsigned short Handle,
    struct sqledinfo **ppDbDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdgne */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdgne (
    unsigned short Handle,
    struct sqledinfo **ppDbDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

Handle
Input. Identifier returned from the associated OPEN DATABASE DIRECTORY SCAN API.

ppDbDirEntry
Output. The caller supplies the API with the address of a pointer to an *sqledinfo* structure. The space for the directory data is allocated by the API, and a pointer to that space is returned to the caller.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
GET DATABASE DIRECTORY ENTRY :scanid [USING :value]
```

REXX API parameters:

scanid A REXX host variable containing the identifier returned from the OPEN DATABASE DIRECTORY SCAN API.

value A compound REXX host variable to which the database entry information is returned. If no name is given, the name SQLDINFO is used. In the following, XXX represents the host variable name (the corresponding field names are taken from the structure returned by the API):

XXX.0	Number of elements in the variable (always 12)

XXX.1	ALIAS (alias of the database)
XXX.2	DBNAME (name of the database)
XXX.3	DRIVE/PATH (local database directory path name)
XXX.3.1	NODE NUMBER (valid for local database directory only)
XXX.4	INTNAME (token identifying the database subdirectory)
XXX.5	NODENAME (name of the node where the database is located)
XXX.6	DBTYPE (product name and release number)
XXX.7	COMMENT (comment associated with the database)
XXX.8	Reserved
XXX.9	TYPE (entry type)
XXX.10	AUTHENTICATION (authentication type)
XXX.10.1	DCE principal
XXX.11	GLBDBNAME (Global database name)
XXX.12	CATALOG NODE NUMBER

Usage notes:

All fields of the directory entry information buffer are padded to the right with blanks.

A subsequent GET NEXT DATABASE DIRECTORY ENTRY obtains the entry following the current entry.

The *sqlcode* value of *sqlca* is set to 1014 if there are no more entries to scan when GET NEXT DATABASE DIRECTORY ENTRY is called.

The count value returned by the OPEN DATABASE DIRECTORY SCAN API can be used to scan through the entire directory by issuing GET NEXT DATABASE DIRECTORY ENTRY calls, one at a time, until the number of scans equals the count of entries.

Related reference:

- “sqledcls - Close Database Directory Scan” on page 330
- “sqledosd - Open Database Directory Scan” on page 334
- “SQLCA” on page 478

- "SQLEDINFO" on page 509

Related samples:

- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"
- "dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqledosd - Open Database Directory Scan

Stores a copy of the system database directory or the local database directory in memory, and returns the number of entries. This copy represents a snapshot of the directory at the time the directory is opened. This copy is not updated, even if the directory itself is changed later.

Use sqledgne - Get Next Database Directory Entry to advance through the database directory, examining information about the database entries. Close the scan using sqledcls - Close Database Directory Scan. This removes the copy of the directory from memory.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledosd */
/* ... */
SQL_API_RC SQL_API_FN
  sqledosd (
    _SQLOLDCHAR *pPath,
    unsigned short *pHandle,
    unsigned short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdosd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdosd (
    unsigned short PathLen,
    struct sqlca *pSqlca,
    unsigned short *pNumEntries,
    unsigned short *pHandle,
    _SQLOLDCHAR *pPath);
/* ... */
```

API parameters:

PathLen
Input. A 2-byte unsigned integer representing the length in bytes of the path parameter. Set to zero if no path is provided.

pSqlca
Output. A pointer to the *sqlca* structure.

pNumEntries
Output. Address of a 2-byte area where the number of directory entries is returned.

pHandle
Output. Address of a 2-byte area for the returned identifier. This identifier must be passed to sqledgne - Get Next Database Directory Entry for scanning the database entries, and to sqledcls - Close Database Directory Scan to release the resources.

pPath Input. The name of the path on which the local database directory resides. If the specified path is a NULL pointer, the system database directory is used.

REXX API syntax:

```
OPEN DATABASE DIRECTORY [ON path_name] USING :value
```

REXX API parameters:

path_name
Name of the path on which the local database directory resides. If the path is not specified, the system database directory is used.

value A compound REXX host variable to which database directory information is returned. In the following, XXX represents the host variable name.

XXX.0	Number of elements in the variable (always 2)
XXX.1	Identifier (handle) for future scan access
XXX.2	Number of entries contained within the directory.

sqledosd - Open Database Directory Scan

Usage notes:

Storage allocated by this API is freed by sqledcls - Close Database Directory Scan.

Multiple OPEN DATABASE DIRECTORY SCAN APIs can be issued against the same directory. However, the results may not be the same. The directory may change between openings.

There can be a maximum of eight opened database directory scans per process.

Related reference:

- "sqledcls - Close Database Directory Scan" on page 330
- "sqledgne - Get Next Database Directory Entry" on page 331
- "SQLCA" on page 478

Related samples:

- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"
- "dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqledpan - Drop Database at Node

Drops a database at a specified database partition server. Can only be run in a partitioned database environment.

Scope:

This API only affects the database partition server on which it is called.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

None. An instance attachment is established for the duration of the call.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledpan */
/* ... */
SQL_API_RC SQL_API_FN
  sqledpan (
    char *pDbAlias,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdpan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdpan (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    void *pReserved2,
    char *pDbAlias);
/* ... */
```

API parameters:

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pSqlca
Output. A pointer to the *sqlca* structure.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

REXX API syntax:

sqledpan - Drop Database at Node

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

This API is used by utilities supplied with DB2 Universal Database Enterprise - Extended Edition, and is not intended for general use. Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqledrpd - Drop Database" on page 340
- "sqlecran - Create Database at Node" on page 312
- "SQLCA" on page 478

sqledreg - Deregister

Deregisters the DB2 server from a network file server. The DB2 server's network address is removed from a specified registry on the file server.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledreg */
/* ... */
SQL_API_RC SQL_API_FN
  sqledreg (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdreg */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdreg (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

Registry
Input. Indicates where on the network file server to deregister the DB2 server. In this release, the only supported registry is `SQL_NWBINDERY` (NetWare file server bindery, defined in `sqlenv`).

pRegisterInfo
Input. A pointer to the *sqle_reg_nwbindery* structure. In this structure, the caller specifies a user name and password that are valid on the network file server.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

When *Registry* has a value of `SQL_NWBINDERY`, this API uses the NetWare user name and password supplied in the *sqle_reg_nwbindery* structure to log onto the NetWare file server (FILESERVER) specified in the database manager configuration file. The object name (OBJECTNAME) specified in the database manager configuration file is deleted from the NetWare file server bindery.

The NetWare user name and password specified must have supervisory or equivalent authority.

This API *must* be issued locally from the DB2 server. It is not supported remotely.

If the IPX/SPX fields are reconfigured, or the DB2 server's IPX/SPX internetwork address changes, deregister the DB2 server from the network file server before making the changes, and then register it again after the changes have been made.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqleregs - Register" on page 380
- "SQLCA" on page 478
- "SQLE-REG-NWBINDERY" on page 501
- "DEREGISTER Command" in the *Command Reference*

sqledrpd - Drop Database

Deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.

Scope:

By default, this API affects all database partition servers that are listed in the `db2nodes.cfg` file.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

Instance. It is not necessary to call ATTACH before dropping a remote database. If the database is cataloged as remote, an instance attachment to the remote node is established for the duration of the call.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledrpd */
/* ... */
SQL_API_RC SQL_API_FN
```

```
  sqledrpd (
    _SQLOLDCHAR *pDbAlias,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdrpd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpd (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pReserved2,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API parameters:

Reserved1
: Reserved for future use.

DbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pSqlca
: Output. A pointer to the *sqlca* structure.

pReserved2
: A spare pointer that is set to null or points to zero. Reserved for future use.

pDbAlias
: Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

REXX API syntax:

```
DROP DATABASE dbalias
```

REXX API parameters:

dbalias
: The alias of the database to be dropped.

Usage notes:

sqledrpd deletes all user data and log files. If the log files are needed for a roll-forward recovery after a restore operation, the files should be saved prior to calling this API.

The database must not be in use; all users must be disconnected from the database before the database can be dropped.

To be dropped, a database must be cataloged in the system database directory. Only the specified database alias is removed from the system database directory. If other aliases with the same database name exist, their entries remain. If the database being dropped is the last entry in the local database directory, the local database directory is deleted automatically.

If this API is called from a remote client (or from a different instance on the same machine), the specified alias is removed from the client's system database directory. The corresponding database name is removed from the server's system database directory.

This API unlinks all files that are linked through any DATALINK columns. Since the unlink operation is performed asynchronously on the DB2 Data Links Manager, its effects may not be seen immediately on the DB2 Data Links Manager, and the unlinked files may not be immediately available for other operations. When the API is called, all the DB2 Data Links Managers configured to that database must be available; otherwise, the drop database operation will fail.

Related reference:
- "sqlecadb - Catalog Database" on page 305
- "sqlecrea - Create Database" on page 314
- "sqleuncd - Uncatalog Database" on page 392
- "sqlecran - Create Database at Node" on page 312
- "sqledpan - Drop Database at Node" on page 336
- "SQLCA" on page 478

Related samples:
- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbcreate.C -- Create and drop databases (C++)"

sqledrpn - Drop Node Verify

Verifies whether a database partition server is being used by a database. A message is returned, indicating whether the database partition server can be dropped.

Scope:

This API only affects the database partition server on which it is issued.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpn (
    unsigned short Action,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpn (
    unsigned short Reserved1,
    struct sqlca *pSqlca,
    void *pReserved2,
    unsigned short Action);
/* ... */
```

API parameters:

Reserved1
Reserved for the length of *pReserved2*.

pSqlca
Output. A pointer to the *sqlca* structure.

pReserved2
A spare pointer that is set to NULL or points to 0. Reserved for future use.

Action
The action requested. The valid value is:

SQL_DROPNODE_VERIFY

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

If a message is returned, indicating that the database partition server is not in use, use the **db2stop** command with DROP NODENUM to remove the entry for the database partition server from the `db2nodes.cfg` file, which removes the database partition server from the partitioned database environment.

If a message is returned, indicating that the database partition server is in use, the following actions should be taken:

1. The database partition server to be dropped will have a database partition on it for each database in the instance. If any of these database partitions contain data, redistribute the database partition groups that use these database partitions. Redistribute the database partition groups to move the data to database partitions that exist at database partition servers that are not being dropped.

   After the database partition groups are redistributed, drop the database partition from every database partition group that uses it. To remove a database partition from a database partition group, you can use either the drop node option of the sqludrdt API or the ALTER DATABASE PARTITION GROUP statement.
2. Drop any event monitors that are defined on the database partition server.
3. Rerun sqledrpn to ensure that the database partition at the database partition server is no longer in use.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqleaddn - Add Node" on page 296

- "SQLCA" on page 478

sqledtin - Detach

Removes the logical instance attachment, and terminates the physical communication connection if there are no other logical connections using this layer.

Authorization:

None

Required connection:

None. Removes an existing instance attachment.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqledtin */
/* ... */
SQL_API_RC SQL_API_FN
  sqledtin (
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgdtin */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdtin (
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
DETACH
```

Related reference:

- "sqleatin - Attach" on page 302
- "SQLCA" on page 478

Related samples:

- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

sqlefmem - Free Memory

Frees memory allocated by DB2 APIs on the caller's behalf. Intended for use with the sqlbtcq and sqlbmtsq APIs.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlefmem */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefmem (
    struct sqlca *pSqlca,
    void *pBuffer);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgfmem */
/* ... */
SQL_API_RC SQL_API_FN
```

```
  sqlgfmem (
    struct sqlca *pSqlca,
    void *pBuffer);
/* ... */
```

API parameters:

pSqlca
: Output. A pointer to the *sqlca* structure.

pBuffer
: Input. Pointer to the memory to be freed.

Related reference:

- "sqlbmtsq - Table Space Query" on page 275
- "sqlbtcq - Table Space Container Query" on page 287
- "SQLCA" on page 478

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"
- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"

sqlefrce - Force Application

Forces local or remote users or applications off the system to allow for maintenance on a server.

Attention: If an operation that cannot be interrupted (RESTORE DATABASE, for example) is forced, the operation must be successfully re-executed before the database becomes available.

Scope:

This API affects all database partition servers that are listed in the `db2nodes.cfg` file.

In a partitioned database environment, this API does not have to be issued from the coordinator partition of the application being forced. This API can be issued from any database partition server in the partitioned database environment.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

Instance. To force users off a remote server, it is necessary to first attach to that server. If no attachment exists, this API is executed locally.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlefrce */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefrce (
    long NumAgentIds,
    sqluint32 *pAgentIds,
    unsigned short ForceMode,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgfrce */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfrce (
    struct sqlca *pSqlca,
    unsigned short ForceMode,
    sqluint32 *pAgentIds,
    long NumAgentIds);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

ForceMode
Input. An integer specifying the operating mode of the FORCE APPLICATION API. Only the asynchronous mode is supported. This means that FORCE APPLICATION does not wait until all specified users are terminated before returning. It returns as soon as the API has been issued successfully, or an error occurs. As a result, there may be a short interval between the time the FORCE APPLICATION call completes and the specified users have been terminated.

This parameter must be set to `SQL_ASYNCH` (defined in `sqlenv`).

pAgentIds
Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding database user.

NumAgentIds
Input. An integer representing the total number of users to be terminated. This number should be the same as the number of elements in the array of agent IDs.

If this parameter is set to `SQL_ALL_USERS` (defined in `sqlenv`), all users are forced. If it is set to zero, an error is returned.

REXX API syntax:

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API parameters:

ALL All applications will be disconnected from their database connection.

agentidarray
A compound REXX host variable containing the list of agent IDs to be terminated. In the following, XXX is the name of the host variable:

XXX.0	Number of agents to be terminated
XXX.1	First agent ID
XXX.2	Second agent ID
XXX.3	and so on.

ASYNC
The only mode currently supported means that FORCE APPLICATION does not wait until all specified applications are terminated before returning.

Usage notes:

db2stop cannot be executed during a force. The database manager remains active so that subsequent database manager operations can be handled without the need for **db2start**.

To preserve database integrity, only users who are idling or executing interruptible database operations can be terminated.

After a FORCE has been issued, the database will still accept requests to connect. Additional forces may be required to completely force all users off.

The database system monitor functions are used to gather the agent IDs of the users to be forced.

When the force mode is set to `SQL_ASYNCH` (the only value permitted), the API immediately returns to the calling application.

Minimal validation is performed on the array of agent IDs to be forced. The user must ensure that the pointer points to an array containing the total number of elements specified. If *NumAgentIds* is set to `SQL_ALL_USERS`, the array is ignored.

When a user is terminated, a ROLLBACK is performed to ensure database consistency.

All users that can be forced will be forced. If one or more specified agent IDs cannot be found, *sqlcode* in the *sqlca* structure is set to 1230. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and sqlefrce is called. The user that calls this API is never forced off.

Agent IDs are recycled, and are used to force applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the wrong user may be forced.

Related reference:
- "db2GetSnapshot - Get Snapshot" on page 73
- "sqleatin - Attach" on page 302
- "sqledtin - Detach" on page 345
- "SQLCA" on page 478

Related samples:
- "dbconn.sqc -- How to connect to and disconnect from a database (C)"
- "dbsample.sqc -- Creates a sample database (C)"

- "instart.c -- Stop and start the current local instance (C)"
- "dbconn.sqC -- How to connect to and disconnect from a database (C++)"
- "instart.C -- Stop and start the current local instance (C++)"
- "dbstop.cbl -- How to stop a database manager (IBM COBOL)"

sqlegdad - Catalog DCS Database

Stores information about remote databases in the Database Connection Services (DCS) directory. These databases are accessed through an Application Requester (AR), such as DB2 Connect. Having a DCS directory entry with a database name matching a database name in the system database directory invokes the specified AR to forward SQL requests to the remote server where the database resides.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegdad */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdad (
    struct sql_dir_entry *pDCSDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggdad */
/* ... */
SQL_API_RC SQL_API_FN
```

```
  sqlggdad (
    struct sqlca *pSqlca,
    struct sql_dir_entry *pDCSDirEntry);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pDCSDirEntry
Input. A pointer to an *sql_dir_entry* (Database Connection Services directory) structure.

REXX API syntax:

```
CATALOG DCS DATABASE dbname [AS target_dbname]
[AR arname] [PARMS parms] [WITH comment]
```

REXX API parameters:

dbname
The local database name of the directory entry to be added.

target_dbname
The target database name.

arname
The application client name.

parms
Parameter string. If specified, the string must be enclosed by double quotation marks.

comment
Description associated with the entry. Maximum length is 30 characters. Enclose the comment by double quotation marks.

Usage notes:

The DB2 Connect program provides connections to DRDA Application Servers such as:
- DB2 for OS/390 databases on System/370 and System/390 architecture host computers
- DB2 for VM and VSE databases on System/370 and System/390 architecture host computers
- OS/400 databases on Application System/400 (AS/400) host computers.

The database manager creates a Database Connection Services directory if one does not exist. This directory is stored on the path that contains the database manager instance that is being used. The DCS directory is maintained outside of the database.

The database must also be cataloged as a remote database in the system database directory.

Note: If directory caching is enabled, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related reference:
- "sqlegdcl - Close DCS Directory Scan" on page 353
- "sqlegdge - Get DCS Directory Entry for Database" on page 357
- "sqlegdgt - Get DCS Directory Entries" on page 359
- "sqlegdsc - Open DCS Directory Scan" on page 361
- "sqlegdel - Uncatalog DCS Database" on page 355
- "SQLCA" on page 478
- "SQL-DIR-ENTRY" on page 467

Related samples:
- "dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlegdcl - Close DCS Directory Scan

Frees the resources that are allocated by "sqlegdsc - Open DCS Directory Scan".

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegdcl */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdcl (
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggdcl */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdcl (
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
CLOSE DCS DIRECTORY
```

Related reference:

- "sqlegdgt - Get DCS Directory Entries" on page 359
- "sqlegdsc - Open DCS Directory Scan" on page 361
- "SQLCA" on page 478

Related samples:

- "dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlegdel - Uncatalog DCS Database

Deletes an entry from the Database Connection Services (DCS) directory.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegdel */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdel (
    struct sql_dir_entry *pDCSDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggdel */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdel (
    struct sqlca *pSqlca,
    struct sql_dir_entry *pDCSDirEntry);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pDCSDirEntry
Input/Output. A pointer to the Database Connection Services directory structure. Fill in the *ldb* field of this structure with the local

name of the database to be deleted. The DCS directory entry with a matching local database name is copied to this structure before being deleted.

REXX API syntax:

```
UNCATALOG DCS DATABASE dbname [USING :value]
```

REXX API parameters:

dbname
The local database name of the directory entry to be deleted.

value A compound REXX host variable into which the directory entry information is returned. In the following, XXX represents the host variable name. If no name is given, the name `SQLGWINF` is used.

XXX.0	Number of elements in the variable (always 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED.

Usage notes:

DCS databases are also cataloged in the system database directory as remote databases that can be uncataloged using the sqleuncd API.

To recatalog a database in the DCS directory, use the sqlegdad API.

To list the DCS databases that are cataloged on a node, use the sqlegdsc, sqlegdgt, and sqlegdcl APIs.

If directory caching is enabled (using the *dir_cache* configuration parameter, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related reference:

- "sqlegdad - Catalog DCS Database" on page 351
- "sqlegdcl - Close DCS Directory Scan" on page 353
- "sqlegdge - Get DCS Directory Entry for Database" on page 357
- "sqlegdgt - Get DCS Directory Entries" on page 359
- "sqlegdsc - Open DCS Directory Scan" on page 361
- "sqleuncd - Uncatalog Database" on page 392
- "SQLCA" on page 478
- "SQL-DIR-ENTRY" on page 467
- "db2CfgGet - Get Configuration Parameters" on page 39

Related samples:

- "dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlegdge - Get DCS Directory Entry for Database

Returns information for a specific entry in the Database Connection Services (DCS) directory.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegdge */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdge (
    struct sql_dir_entry *pDCSDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

sqlegdge - Get DCS Directory Entry for Database

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggdge */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdge (
    struct sqlca *pSqlca,
    struct sql_dir_entry *pDCSDirEntry);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pDCSDirEntry
Input/Output. Pointer to the Database Connection Services directory structure. Fill in the *ldb* field of this structure with the local name of the database whose DCS directory entry is to be retrieved. The remaining fields in the structure are filled in upon return of this API.

REXX API syntax:

```
GET DCS DIRECTORY ENTRY FOR DATABASE dbname [USING :value]
```

REXX API parameters:

dbname
Specifies the local database name of the directory entry to be obtained.

value A compound REXX host variable into which the directory entry information is returned. In the following, XXX represents the host variable name. If no name is given, the name `SQLGWINF` is used.

XXX.0	Number of elements in the variable (always 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED.

Related reference:

- "sqlegdad - Catalog DCS Database" on page 351
- "sqlegdcl - Close DCS Directory Scan" on page 353

- "sqlegdgt - Get DCS Directory Entries" on page 359
- "sqlegdsc - Open DCS Directory Scan" on page 361
- "sqlegdel - Uncatalog DCS Database" on page 355
- "SQL-DIR-ENTRY" on page 467

Related samples:

- "dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlegdgt - Get DCS Directory Entries

Transfers a copy of Database Connection Services (DCS) directory entries to a buffer supplied by the application.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegdgt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdgt (
    short *pNumEntries,
    struct sql_dir_entry *pDCSDirEntries,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggdgt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdgt (
    struct sqlca *pSqlca,
    short *pNumEntries,
    struct sql_dir_entry *pDCSDirEntries);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pNumEntries
Input/Output. Pointer to a short integer representing the number of entries to be copied to the caller's buffer. The number of entries actually copied is returned.

pDCSDirEntries
Output. Pointer to a buffer where the collected DCS directory entries will be held upon return of the API call. The buffer must be large enough to hold the number of entries specified in the *pNumEntries* parameter.

REXX API syntax:

```
GET DCS DIRECTORY ENTRY [USING :value]
```

REXX API parameters:

value A compound REXX host variable into which the directory entry information is returned. In the following, XXX represents the host variable name. If no name is given, the name SQLGWINF is used.

XXX.0	Number of elements in the variable (always 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED.

Usage notes:

sqlegdsc - Open DCS Directory Scan, which returns the entry count, must be called prior to issuing GET DCS DIRECTORY ENTRIES.

If all entries are copied to the caller, the Database Connection Services directory scan is automatically closed, and all resources are released.

If entries remain, subsequent calls to this API should be made, or CLOSE DCS DIRECTORY SCAN should be called, to release system resources.

Related reference:

- "sqlegdcl - Close DCS Directory Scan" on page 353
- "sqlegdge - Get DCS Directory Entry for Database" on page 357
- "sqlegdsc - Open DCS Directory Scan" on page 361
- "SQLCA" on page 478
- "SQL-DIR-ENTRY" on page 467

Related samples:

- "dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlegdsc - Open DCS Directory Scan

Stores a copy in memory of the Database Connection Services directory entries, and returns the number of entries. This is a snapshot of the directory at the time the directory is opened.

The copy is not updated if the directory itself changes after a call to this API. Use sqlegdgt - Get DCS Directory Entries to retrieve the entries, and sqlegdcl - Close DCS Directory Scan to release the resources associated with calling this API.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegdsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdsc (
    short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggdsc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdsc (
    struct sqlca *pSqlca,
    short *pNumEntries);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pNumEntries
Output. Address of a 2-byte area to which the number of directory entries is returned.

REXX API syntax:

```
OPEN DCS DIRECTORY
```

Usage notes:

The caller of the scan uses the returned value *pNumEntries* to allocate enough memory to receive the entries. If a scan call is received while a copy is already held, the previous copy is released, and a new copy is collected.

Related reference:

- "sqlegdcl - Close DCS Directory Scan" on page 353
- "sqlegdge - Get DCS Directory Entry for Database" on page 357
- "sqlegdgt - Get DCS Directory Entries" on page 359
- "SQLCA" on page 478

Related samples:

- "dcscat.cbl -- Get information for a DCS directory in a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"

- “ininfo.C -- Set and get information at the instance level (C++)”

sqlegins - Get Instance

Returns the value of the **DB2INSTANCE** environment variable.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlegins */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegins (
    _SQLOLDCHAR *pInstance,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlggins */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggins (
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pInstance);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

pInstance
Output. Pointer to a string buffer where the database manager instance name is placed. This buffer must be at least 8 bytes in length.

REXX API syntax:

sqlegins - Get Instance

```
GET INSTANCE INTO :instance
```

REXX API parameters:

instance
A REXX host variable into which the database manager instance name is to be placed.

Usage notes:

The value in the **DB2INSTANCE** environment variable is not necessarily the instance to which the user is attached.

To identify the instance to which a user is currently attached, call sqleatin - Attach, with null arguments except for the *sqlca* structure.

Related reference:

- "sqleatin - Attach" on page 302
- "SQLCA" on page 478

Related samples:

- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqleintr - Interrupt

Stops a request. This API is called from a control break signal handler in an application. The control break signal handler can be the default, installed by sqleisig - Install Signal Handler, or a routine supplied by the programmer and installed using an appropriate operating system call.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleintr */
/* ... */
SQL_API_RC SQL_API_FN
  sqleintr (
    void);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgintr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintr (
    void);
/* ... */
```

API parameters:

None

REXX API syntax:

```
INTERRUPT
```

Examples:

```
   call SQLDBS 'INTERRUPT'
```

Usage notes:

No database manager APIs should be called from an interrupt handler except **sqleintr**. However, the system will not prevent it.

Any database transaction in a state of committing or rollback cannot be interrupted.

An interrupted database manager request returns a code indicating that it was interrupted.

The following table summarizes the effect of an interrupt operation on other APIs:

Table 7. INTERRUPT Actions

Database Activity	Action
BACKUP	Utility cancelled. Data on media may be incomplete.
BIND	Binding cancelled. Package creation rolled back.

Table 7. INTERRUPT Actions (continued)

Database Activity	Action
COMMIT	None. COMMIT completes.
CREATE DATABASE/CREATE DATABASE AT NODE/ADD NODE/DROP NODE VERIFY	After a certain point, these APIs are not interruptible. If the interrupt call is received before this point, the database is not created. If the interrupt call is received after this point, it is ignored.
DROP DATABASE/DROP DATABASE AT NODE	None. The APIs complete.
EXPORT/IMPORT/RUNSTATS	Utility cancelled. Database updates rolled back.
FORCE APPLICATION	None. FORCE APPLICATION completes.
LOAD	Utility cancelled. Data in table may be incomplete.
PREP	Precompile cancelled. Package creation rolled back.
REORGANIZE TABLE	The interrupt will be delayed until the copy is complete. The recreation of the indexes will be resume on the next attempt to access the table.
RESTORE	Utility cancelled. DROP DATABASE performed. Not applicable to table space level restore.
ROLLBACK	None. ROLLBACK completes.
Directory Services	Directory left in consistent state. Utility function may or may not be performed.
SQL Data Definition statements	Database transactions are set to the state existing prior to invocation of the SQL statement.
Other SQL statements	Database transactions are set to the state existing prior to invocation of the SQL statement.

sqleisig - Install Signal Handler

Installs the default interrupt (usually Control-C and/or Control-Break) signal handler. When this default handler detects an interrupt signal, it resets the signal and calls sqleintr.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleisig */
/* ... */
SQL_API_RC SQL_API_FN
  sqleisig (
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgisig */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgisig (
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
INSTALL SIGNAL HANDLER
```

Usage notes:

If an application has no signal handler, and an interrupt is received, the application is terminated. This API provides simple signal handling, and can be used if an application does not have extensive interrupt handling requirements.

The API must be called for the interrupt signal handler to function properly.

If an application requires a more elaborate interrupt handling scheme, a signal handling routine that can also call the sqleintr API can be developed. Use either the operating system call or the language-specific library signal function. The sqleintr API should be the only database manager operation performed by a customized signal handler. Follow all operating system programming techniques and practices to ensure that the previously installed signal handlers work properly.

Related reference:

- "sqleintr - Interrupt" on page 364
- "SQLCA" on page 478

Related samples:

- "dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)"

sqlemgdb - Migrate Database

Converts previous (Version 2.x or higher) versions of DB2 databases to current formats.

Authorization:

sysadm

Required connection:

This API establishes a database connection.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlemgdb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlemgdb (
    _SQLOLDCHAR *pDbAlias,
    _SQLOLDCHAR *pUserName,
    _SQLOLDCHAR *pPassword,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgmgdb */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmgdb (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pPassword,
    _SQLOLDCHAR *pUserName,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API parameters:

PasswordLen
: Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero when no password is supplied.

UserNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero when no user name is supplied.

DbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pSqlca
: Output. A pointer to the *sqlca* structure.

pPassword
: Input. A string containing the password of the supplied user name (if any). May be NULL.

pUserName
: Input. A string containing the user name of the application. May be NULL.

pDbAlias
: Input. A string containing the alias of the database that is cataloged in the system database directory.

REXX API syntax:

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API parameters:

dbalias
: Alias of the database to be migrated.

username
: User name under which the database is to be restarted.

password
: Password used to authenticate the user name.

Usage notes:

sqlemgdb - Migrate Database

This API will only migrate a database to a newer version, and cannot be used to convert a migrated database to its previous version.

The database must be cataloged before migration.

Related reference:

- "SQLCA" on page 478

Related samples:

- "dbmigrat.c -- Migrate a database (C)"
- "dbmigrat.C -- Migrate a database (C++)"
- "migrate.cbl -- Demonstrates how to migrate to a database (IBM COBOL)"

sqlencls - Close Node Directory Scan

Frees the resources that are allocated by "sqlenops - Open Node Directory".

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlencls */
/* ... */
SQL_API_RC SQL_API_FN
  sqlencls (
    unsigned short Handle,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgncls */
/* ... */
SQL_API_RC SQL_API_FN
```

```
  sqlgncls (
    unsigned short Handle,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

Handle
Input. Identifier returned from the associated OPEN NODE DIRECTORY SCAN API.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
CLOSE NODE DIRECTORY :scanid
```

REXX API parameters:

scanid A host variable containing the *scanid* returned from the OPEN NODE DIRECTORY SCAN API.

Related reference:

- "sqlengne - Get Next Node Directory Entry" on page 371
- "sqlenops - Open Node Directory Scan" on page 374
- "SQLCA" on page 478

Related samples:

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"
- "nodecat.cbl -- Get node directory information (IBM COBOL)"

sqlengne - Get Next Node Directory Entry

Returns the next entry in the node directory after sqlenops - Open Node Directory Scan" is called. Subsequent calls to this API return additional entries.

Authorization:

None

Required connection:

None

sqlengne - Get Next Node Directory Entry

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlengne */
/* ... */
SQL_API_RC SQL_API_FN
  sqlengne (
    unsigned short Handle,
    struct sqleninfo **ppNodeDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgngne */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgngne (
    unsigned short Handle,
    struct sqleninfo **ppNodeDirEntry,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

Handle
Input. Identifier returned from sqlenops - Open Node Directory Scan.

ppNodeDirEntry
Output. Address of a pointer to an *sqleninfo* structure. The caller of this API does not have to provide memory for the structure, just the pointer. Upon return from the API, the pointer points to the next node directory entry in the copy of the node directory allocated by sqlenops - Open Node Directory Scan.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
GET NODE DIRECTORY ENTRY :scanid [USING :value]
```

REXX API parameters:

scanid A REXX host variable containing the identifier returned from the OPEN NODE DIRECTORY SCAN API.

value A compound REXX host variable to which the node entry information is returned. If no name is given, the name SQLNINFO is used. In the following, XXX represents the host variable name (the corresponding field names are taken from the structure returned by the API):

XXX.0	Number of elements in the variable (always 16)
XXX.1	NODENAME
XXX.2	LOCALLU
XXX.3	PARTNERLU
XXX.4	MODE
XXX.5	COMMENT
XXX.6	RESERVED
XXX.7	PROTOCOL (protocol type)
XXX.8	ADAPTER (NetBIOS adapter #)
XXX.9	RESERVED
XXX.10	SYMDESTNAME (symbolic destination name)
XXX.11	SECURITY (security type)
XXX.12	HOSTNAME
XXX.13	SERVICENAME
XXX.14	FILESERVER
XXX.15	OBJECTNAME
XXX.16	INSTANCE (local instance name).

Usage notes:

All fields in the node directory entry information buffer are padded to the right with blanks.

The *sqlcode* value of *sqlca* is set to 1014 if there are no more entries to scan when this API is called.

The entire directory can be scanned by calling this API *pNumEntries* times.

Related reference:

- "sqlencls - Close Node Directory Scan" on page 370
- "sqlenops - Open Node Directory Scan" on page 374
- "SQLCA" on page 478
- "SQLENINFO" on page 512

sqlengne - Get Next Node Directory Entry

Related samples:

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"
- "nodecat.cbl -- Get node directory information (IBM COBOL)"

sqlenops - Open Node Directory Scan

Stores a copy in memory of the node directory, and returns the number of entries. This is a snapshot of the directory at the time the directory is opened. This copy is not updated, even if the directory itself is changed later.

Use sqlengne - Get Next Node Directory Entry to advance through the node directory and examine information about the node entries. Close the scan using sqlencls - Close Node Directory Scan. This removes the copy of the directory from memory.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlenops */
/* ... */
SQL_API_RC SQL_API_FN
  sqlenops (
    unsigned short *pHandle,
    unsigned short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgnops */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgnops (
    unsigned short *pHandle,
    unsigned short *pNumEntries,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pHandle
: Output. Identifier returned from this API. This identifier must be passed to sqlengne - Get Next Node Directory Entry, and sqlencls - Close Node Directory Scan.

pNumEntries
: Output. Address of a 2-byte area to which the number of directory entries is returned.

pSqlca
: Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
OPEN NODE DIRECTORY USING :value
```

REXX API parameters:

value A compound REXX variable to which node directory information is returned. In the following, XXX represents the host variable name.

XXX.0	Number of elements in the variable (always 2)
XXX.1	Specifies a REXX host variable containing a number for *scanid*
XXX.2	The number of entries contained within the directory.

Usage notes:

Storage allocated by this API is freed by calling sqlencls - Close Node Directory Scan.

Multiple node directory scans can be issued against the node directory. However, the results may not be the same. The directory may change between openings.

There can be a maximum of eight node directory scans per process.

Related reference:
- "sqlencls - Close Node Directory Scan" on page 370
- "sqlengne - Get Next Node Directory Entry" on page 371
- "SQLCA" on page 478

Related samples:

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"
- "nodecat.cbl -- Get node directory information (IBM COBOL)"

sqleqryc - Query Client

Returns current connection settings for an application process.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleqryc */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgqryc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgqryc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pConnectionSettings
Input/Output. A pointer to an *sqle_conn_setting* structure, which

specifies connection setting types and values. The user defines an array of *NumSettings* connection settings structures, and sets the *type* field of each element in this array to indicate one of the five possible connection settings options. Upon return, the *value* field of each element contains the current setting of the option specified.

NumSettings
Input. Any integer (from 0 to 7) representing the number of connection option values to be returned.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
QUERY CLIENT INTO :output
```

REXX API parameters:

output
A compound REXX host variable containing information about the current connection settings of the application process. In the following, XXX represents the host variable name.

XXX.1	Current connection setting for the CONNECTION type
XXX.2	Current connection setting for the SQLRULES
XXX.3	Current connection setting indicating which connections will be released when a COMMIT is issued.
XXX.4	Current connection setting of the SYNCPOINT option. Indicates whether a transaction manager should be used to enforce two-phase commit semantics, whether the database manager should ensure that there is only one database being updated when multiple databases are accessed within a single transaction, or whether neither of these options is to be used.
XXX.5	Current connection setting for the maximum number of concurrent connections for a NETBIOS adapter.
XXX.6	Current connection setting for deferred PREPARE.

Usage notes:

The connection settings for an application process can be queried at any time during execution.

If QUERY CLIENT is successful, the fields in the *sqle_conn_setting* structure will contain the current connection settings of the application process. If SET CLIENT has never been called, the settings will contain the values of the precompile options only if an SQL statement has already been processed; otherwise, they will contain the default values for the precompile options.

Related reference:

- "sqlesetc - Set Client" on page 386
- "sqleqryi - Query Client Information" on page 378
- "SQLCA" on page 478
- "SQLE-CONN-SETTING" on page 489

Related samples:

- "cli_info.c -- Set and get information at the client level (C)"
- "cli_info.C -- Set and get information at the client level (C++)"
- "client.cbl -- How to set and query a client (IBM COBOL)"

sqleqryi - Query Client Information

Returns existing client information. Since this API permits specification of a database alias, an application can query client information associated with a specific connection. Returns null if the sqleseti API has not previously established a value.

If a specific connection is requested, this API returns the latest values for that connection. If all connections are specified, the API returns the values that are to be associated with all connections; that is, the values passed in the last call to sqleseti (specifying all connections).

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleqryi */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryi (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqleqryi */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryi (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

DbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database alias. If a value greater than zero is provided, *pDbAlias* must point to the alias name. Returns the settings associated with the last call to sqleseti for this alias (or a call to sqleseti specifying a zero length alias). If zero is specified, returns the settings associated with the last call to sqleseti which specified a zero length alias.

pDbAlias
: Input. A pointer to a string containing the database alias.

NumItems
: Input. Number of entries being modified. The minimum value is 1.

pClient_Info
: Input. A pointer to an array of *NumItems sqle_client_info* structures, each containing a type field indicating which value to return, and a pointer to the returned value. The area pointed to must be large enough to accommodate the value being requested.

pSqlca
: Output. A pointer to the *sqlca* structure.

Usage notes:

The settings can be queried at any time during execution. If the API call is successful, the current settings are returned to the specified areas. Returns a length of zero and a null-terminated string (\0) for any fields that have not been set through a call to the sqleseti API.

Related reference:

- "sqleseti - Set Client Information" on page 389
- "SQLCA" on page 478
- "SQLE-CLIENT-INFO" on page 486

Related samples:

- "cli_info.c -- Set and get information at the client level (C)"
- "cli_info.C -- Set and get information at the client level (C++)"

sqleregs - Register

Registers the DB2 server on the network server. The DB2 server's network address is stored in a specified registry on the file server, where it can be retrieved by a client application that uses the IPX/SPX communication protocol.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleregs */
/* ... */
SQL_API_RC SQL_API_FN
  sqleregs (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgregs */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgregs (
    unsigned short Registry,
    void *pRegisterInfo,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

Registry
Input. Indicates where on the network file server to register the DB2 server. In this release, the only supported value is `SQL_NWBINDERY` (NetWare file server bindery, defined in `sqlenv`).

pRegisterInfo
Input. A pointer to the *sqle_reg_nwbindery* structure. In the structure, the caller specifies a user name and password that are valid on the network file server.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

This API determines the IPX/SPX address of the DB2 server machine (the machine from which it was called), and then creates an object in the NetWare file server bindery using the value for *objectname* specified in the database manager configuration file. The IPX/SPX address of the DB2 server is stored as a property in that object. In order for a client to connect or attach to a DB2 database using IPX/SPX file server addressing, it must catalog an IPX/SPX node (using the same FILESERVER and OBJECTNAME specified on the server) in the node directory.

The specified NetWare user name and password must have supervisory or equivalent authority.

This API *must* be issued locally from a DB2 server. It is not supported remotely.

After installation and configuration of DB2, the DB2 server should be registered once on the network file server (unless only *direct addressing* will be used by IPX/SPX clients to connect to this DB2 server). After that, if the

IPX/SPX fields are reconfigured, or the DB2 server's IPX/SPX internetwork address changes, deregister the DB2 server on the network file server before making the changes, and then register it again after the changes have been made.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "sqledreg - Deregister" on page 338
- "SQLCA" on page 478
- "SQLE-REG-NWBINDERY" on page 501
- "REGISTER Command" in the *Command Reference*

sqlesact - Set Accounting String

Provides accounting information that will be sent to a DRDA server with the application's next connect request.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlesact */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesact (
    char *pAccountingString,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgsact */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsact (
    unsigned short AccountingStringLen,
    char *pAccountingString,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

AccountingStringLen
Input. A 2-byte unsigned integer representing the length in bytes of the accounting string.

pAccountingString
Input. A string containing the accounting data.

pSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

To send accounting data with a connect request, an application should call this API before connecting to a database. The accounting string can be changed before connecting to another database by calling the API again; otherwise, the value remains in effect until the end of the application. The accounting string can be at most `SQL_ACCOUNT_STR_SZ` (defined in `sqlenv`) bytes long; longer strings will be truncated. To ensure that the accounting string is converted correctly when transmitted to the DRDA server, use only the characters A to Z, 0 to 9, and the underscore (_).

Related reference:

- "sqleseti - Set Client Information" on page 389
- "SQLCA" on page 478

Related samples:

- "cli_info.c -- Set and get information at the client level (C)"
- "cli_info.C -- Set and get information at the client level (C++)"
- "setact.cbl -- How to set accounting string (IBM COBOL)"

sqlesdeg - Set Runtime Degree

Sets the maximum run time degree of intra-partition parallelism for SQL statements for specified active applications. It has no effect on CREATE INDEX parallelism.

Scope:

This API affects all database partition servers that are listed in the `db2nodes.cfg` file.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

Instance. To change the maximum run time degree of parallelism on a remote server, it is first necessary to attach to that server. If no attachment exists, the SET RUNTIME DEGREE statement fails.

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlesdeg */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesdeg (
    sqlint32 NumAgentIds,
    sqluint32 *pAgentIds,
    sqlint32 Degree,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgsdeg */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsdeg (
    struct sqlca *pSqlca,
    sqlint32 Degree,
    sqluint32 *pAgentIds,
    sqlint32 NumAgentIds);
/* ... */
```

API parameters:

pSqlca
: Output. A pointer to the *sqlca* structure.

Degree
: Input. The new value for the maximum run time degree of parallelism. The value must be in the range 1 to 32767.

pAgentIds
: Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding application. To list the agent IDs of the active applications, use the db2GetSnapshot API.

NumAgentIds
: Input. An integer representing the total number of active applications to which the new degree value will apply. This number should be the same as the number of elements in the array of agent IDs.

  If this parameter is set to `SQL_ALL_USERS` (defined in `sqlenv`), the new degree will apply to all active applications. If it is set to zero, an error is returned.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

The database system monitor functions are used to gather the agent IDs and degrees of active applications.

Minimal validation is performed on the array of agent IDs. The user must ensure that the pointer points to an array containing the total number of elements specified. If *NumAgentIds* is set to `SQL_ALL_USERS`, the array is ignored.

If one or more specified agent IDs cannot be found, the unknown agent IDs are ignored, and the function continues. No error is returned. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and the API is called.

Agent IDs are recycled, and are used to change the degree of parallelism for applications some time after being gathered by the database system monitor.

When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the new degree of parallelism will be modified for the wrong user.

Related tasks:

- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "db2GetSnapshot - Get Snapshot" on page 73
- "SQLCA" on page 478
- "SET RUNTIME DEGREE Command" in the *Command Reference*

Related samples:

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlesetc - Set Client

Specifies connection settings for the application.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqlesetc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesetc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlgsetc */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsetc (
    struct sqle_conn_setting *pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pConnectionSettings
Input. A pointer to the *sqle_conn_setting* structure, which specifies connection setting types and values. Allocate an array of *NumSettings* *sqle_conn_setting* structures. Set the *type* field of each element in this array to indicate the connection option to set. Set the *value* field to the desired value for the option.

NumSettings
Input. Any integer (from 0 to 7) representing the number of connection option values to set.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
SET CLIENT USING :values
```

REXX API parameters:

values A compound REXX host variable containing the connection settings for the application process. In the following, XXX represents the host variable name.

XXX.0 Number of connection settings to be established

XXX.1 Specifies how to set up the CONNECTION type. The valid values are:

1 Type 1 CONNECT

2 Type 2 CONNECT

XXX.2 Specifies how to set up the SQLRULES. The valid values are:

DB2 Process type 2 CONNECT according to the DB2 rules

STD Process type 2 CONNECT according to the Standard rules

XXX.3 Specifies how to set up the scope of disconnection to databases at commit. The valid values are:

EXPLICIT	Disconnect only those marked by the SQL RELEASE statement
CONDITIONAL	Disconnect only those that have no open WITH HOLD cursors
AUTOMATIC	Disconnect all connections

XXX.4 Specifies how to set up the coordination among multiple database connections during commits or rollbacks. The valid values are:

TWOPHASE	Use Transaction Manager (TM) to coordinate two-phase commits

XXX.5 Specifies the maximum number of concurrent connections for a NETBIOS adapter.

XXX.6 Specifies when to execute the PREPARE statement. The valid values are:

NO	The PREPARE statement will be executed at the time it is issued
YES	The PREPARE statement will not be executed until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued. However, the PREPARE INTO statement is not deferred
ALL	Same as YES, except that the PREPARE INTO statement is also deferred

Usage notes:

If this API is successful, the connections in the subsequent units of work will use the connection settings specified. If this API is unsuccessful, the connection settings are unchanged.

The connection settings for the application can only be changed when there are no existing connections (for example, before any connection is established, or after RELEASE ALL and COMMIT).

Once the SET CLIENT API has executed successfully, the connection settings are fixed and can only be changed by again executing the SET CLIENT API. All corresponding precompiled options of the application modules will be overridden.

Related reference:

- “sqleqryc - Query Client” on page 376

- "sqleseti - Set Client Information" on page 389
- "SQLCA" on page 478
- "SQLE-CONN-SETTING" on page 489

Related samples:

- "cli_info.c -- Set and get information at the client level (C)"
- "dbcfg.sqc -- Configure database and database manager configuration parameters (C)"
- "dbmcon.sqc -- How to use multiple databases (C)"
- "cli_info.C -- Set and get information at the client level (C++)"
- "dbcfg.sqC -- Configure database and database manager configuration parameters (C++)"
- "dbmcon.sqC -- How to use multiple databases (C++)"
- "client.cbl -- How to set and query a client (IBM COBOL)"

sqleseti - Set Client Information

Permits an application to set client information associated with a specific connection, provided a connection already exists.

In a TP monitor or 3-tier client/server application environment, there is a need to obtain information about the client, and not just the application server that is working on behalf of the client. By using this API, the application server can pass the client's user ID, workstation information, program information, and other accounting information to the DB2 server; otherwise, only the application server's information is passed, and that information is likely to be the same for the many client invocations that go through the same application server.

The application can elect to not specify an alias, in which case the client information will be set for all existing, as well as future, connections. This API will only permit information to be changed outside of a unit of work, either before any SQL is executed, or after a commit or a rollback. If the call is successful, the values for the connection will be sent at the next opportunity, grouped with the next SQL request sent on that connection; a successful call means that the values have been accepted, and that they will be propagated to subsequent connections.

This API can be used to establish values prior to connecting to a database, or it can be used to set or modify the values once a connection has been established.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleseti */
/* ... */
SQL_API_RC SQL_API_FN
  sqleseti (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqleseti */
/* ... */
SQL_API_RC SQL_API_FN
  sqleseti (
    unsigned short DbAliasLen,
    char *pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info*pClient_Info,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias. If a value greater than zero is provided, *pDbAlias* must point to the alias name, and the settings will affect only the specified connection. If zero is specified, the settings will affect all existing and future connections.

pDbAlias
Input. A pointer to a string containing the database alias.

NumItems
Input. Number of entries being modified. The minimum value is 1.

pClient_Info
Input. A pointer to an array of *NumItems sqle_client_info* structures, each containing a type field indicating which value to set, the length of that value, and a pointer to the new value.

pSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

If an alias name was provided, a connection to the alias must already exist, and all connections to that alias will inherit the changes. The information will be retained until the connection for that alias is broken. If an alias name was not provided, settings for all existing connections will be changed, and any future connections will inherit the changes. The information will be retained until the program terminates.

The field names represent guidelines for the type of information that can be provided. For example, a TP monitor application could choose to provide the TP monitor transaction ID along with the application name in the SQL_CLIENT_INFO_APPLNAM field. This would provide better monitoring and accounting on the DB2 server, where the DB2 transaction ID can be associated with the TP monitor transaction ID.

Currently this API will pass information to DB2 OS/390 Version 5 and higher and DB2 UDB Version 7 and higher. All information (except the accounting string) is displayed on the DISPLAY THREAD command, and will all be logged into the accounting records.

The data values provided with the API can also be accessed by SQL special register. The values in these registers are stored in the database code page. Data values provided with this API are converted to the database code page before being stored in the special registers. Any data value that exceeds the maximum supported size after conversion to the database code page will be truncated before being stored at the server. These truncated values will be returned by the special registers. The original data values will also be stored at the server and are not converted to the database code page. The unconverted values can be returned by calling the sqleqryi API.

Related reference:
- "db2GetSnapshot - Get Snapshot" on page 73
- "sqlesetc - Set Client" on page 386
- "sqlesact - Set Accounting String" on page 382
- "sqleqryi - Query Client Information" on page 378
- "SQLCA" on page 478

- "SQLE-CLIENT-INFO" on page 486

Related samples:

- "cli_info.c -- Set and get information at the client level (C)"
- "cli_info.C -- Set and get information at the client level (C++)"

sqleuncd - Uncatalog Database

Deletes an entry from the system database directory.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleuncd */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncd (
    _SQLOLDCHAR *pDbAlias,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlguncd */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncd (
    unsigned short DbAliasLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pDbAlias);
/* ... */
```

API parameters:

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pSqlca
Output. A pointer to the *sqlca* structure.

pDbAlias
Input. A string containing the database alias that is to be uncataloged.

REXX API syntax:

```
UNCATALOG DATABASE dbname
```

REXX API parameters:

dbname
Alias of the database to be uncataloged.

Usage notes:

Only entries in the system database directory can be uncataloged. Entries in the local database directory can be deleted using the sqledrpd API.

To recatalog the database, use the sqlecadb API.

To list the databases that are cataloged on a node, use the sqledosd, sqledgne, and sqledcls APIs.

The authentication type of a database, used when communicating with a down-level server, can be changed by first uncataloging the database, and then cataloging it again with a different type.

If directory caching is enabled using the *dir_cache* configuration parameter, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related reference:
- "sqlecadb - Catalog Database" on page 305
- "sqledcls - Close Database Directory Scan" on page 330
- "sqledrpd - Drop Database" on page 340
- "sqledgne - Get Next Database Directory Entry" on page 331
- "sqledosd - Open Database Directory Scan" on page 334
- "SQLCA" on page 478
- "db2CfgGet - Get Configuration Parameters" on page 39

Related samples:
- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqleuncn - Uncatalog Node

Deletes an entry from the node directory.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

API include file:

sqlenv.h

C API syntax:

```
/* File: sqlenv.h */
/* API: sqleuncn */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncn (
    _SQLOLDCHAR *pNodeName,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlenv.h */
/* API: sqlguncn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncn (
    unsigned short NodeNameLen,
    struct sqlca *pSqlca,
    _SQLOLDCHAR *pNodeName);
/* ... */
```

API parameters:

NodeNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the node name.

pSqlca
: Output. A pointer to the *sqlca* structure.

pNodeName
: Input. A string containing the name of the node to be uncataloged.

REXX API syntax:

```
UNCATALOG NODE nodename
```

REXX API parameters:

nodename
: Name of the node to be uncataloged.

Usage notes:

To recatalog the node, use the sqlectnd API.

To list the nodes that are cataloged, use the sqledosd, sqledgne, and sqledcls APIs.

If directory caching is enabled using the *dir_cache* configuration parameter, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related reference:

- "sqlectnd - Catalog Node" on page 322
- "sqlencls - Close Node Directory Scan" on page 370

- “sqlengne - Get Next Node Directory Entry” on page 371
- “sqlenops - Open Node Directory Scan” on page 374
- “SQLCA” on page 478
- “db2CfgGet - Get Configuration Parameters” on page 39

Related samples:
- “ininfo.c -- Set and get information at the instance level (C)”
- “ininfo.C -- Set and get information at the instance level (C++)”
- “nodecat.cbl -- Get node directory information (IBM COBOL)”

sqlgaddr - Get Address

Places the address of a variable into another variable. It is used in host languages, such as FORTRAN and COBOL, that do not provide pointer manipulation.

Authorization:

None

Required connection:

None

API include file:

sqlutil.h

Generic API syntax:
```
/* File: sqlutil.h */
/* API: sqlgaddr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddr (
    char *pVariable,
    char **ppOutputAddress);
/* ... */
```

API parameters:

pVariable
Input. Variable whose address is to be returned.

ppOutputAddress
Output. A 4-byte area into which the variable address is returned.

Usage notes:

This API is used in the COBOL and FORTRAN languages only.

Related reference:

- "sqlgdref - Dereference Address" on page 397

sqlgdref - Dereference Address

Copies data from a buffer that is defined by a pointer, into a variable that is directly accessible by the application. It is used in host languages, such as FORTRAN and COBOL, that do not provide pointer manipulation. This API can be used to obtain results from APIs that return a pointer to the desired data.

Authorization:

None

Required connection:

None

API include file:

sqlutil.h

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgdref */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdref (
    unsigned int NumBytes,
    char *pTargetBuffer,
    char **ppSourceBuffer);
/* ... */
```

API parameters:

NumBytes
Input. An integer representing the number of bytes to be transferred.

pTargetBuffer
Output. Area into which the data are moved.

ppSourceBuffer
Input. A pointer to the area containing the desired data.

Usage notes:

This API is used in the COBOL and FORTRAN languages only.

Related reference:

- "sqlgaddr - Get Address" on page 396

sqlgmcpy - Copy Memory

Copies data from one memory area to another. It is used in host languages, such as FORTRAN and COBOL, that do not provide memory block copy functions.

Authorization:

None

Required connection:

None

API include file:

sqlutil.h

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgmcpy */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmcpy (
    void *pTargetBuffer,
    const void *pSource,
    sqluint32 NumBytes);
/* ... */
```

API parameters:

pTargetBuffer
Output. Area into which to move the data.

pSource
Input. Area from which to move the data.

NumBytes
Input. A 4-byte unsigned integer representing the number of bytes to be transferred.

Usage notes:

This API is used in the COBOL and FORTRAN languages only.

Related reference:
- "sqlgaddr - Get Address" on page 396

sqlogstt - Get SQLSTATE Message

Retrieves the message text associated with an SQLSTATE.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
/* File: sql.h */
/* API: sqlogstt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlogstt (
    char *pBuffer,
    short BufferSize,
    short LineWidth,
    char *pSqlstate);
/* ... */
```

Generic API syntax:

```
/* File: sql.h */
/* API: sqlggstt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggstt (
    short BufferSize,
    short LineWidth,
    char *pSqlstate,
    char *pBuffer);
/* ... */
```

API parameters:

BufferSize
: Input. Size, in bytes, of a string buffer to hold the retrieved message text.

LineWidth
: Input. The maximum line width for each line of message text. Lines are broken on word boundaries. A value of zero indicates that the message text is returned without line breaks.

pSqlstate
: Input. A string containing the SQLSTATE for which the message text is to be retrieved. This field is alphanumeric and must be either five-digit (specific SQLSTATE) or two-digit (SQLSTATE class, first two digits of an SQLSTATE). This field does not need to be NULL-terminated if 5 digits are being passed in, but must be NULL-terminated if 2 digits are being passed.

pBuffer
: Output. A pointer to a string buffer where the message text is to be placed. If the message must be truncated to fit in the buffer, the truncation allows for the null string terminator character.

REXX API syntax:

```
GET MESSAGE FOR SQLSTATE sqlstate INTO :msg [LINEWIDTH width]
```

REXX API parameters:

sqlstate
: The SQLSTATE for which the message text is to be retrieved.

msg REXX variable into which the message is placed.

width Maximum line width for each line of the message text. The line is broken on word boundaries. If a value is not specified, or this parameter is set to 0, the message text returns without line breaks.

Usage notes:

One message is returned per call.

A LF/NULL sequence is placed at the end of each message.

If a positive line width is specified, LF/NULL sequences are inserted between words so that the lines do not exceed the line width.

If a word is longer than a line width, the line is filled with as many characters as will fit, a LF/NULL is inserted, and the remaining characters are placed on the next line.

Return codes:

Code	Message
+i	Positive integer indicating the number of bytes in the formatted message. If this is greater than the buffer size input by the caller, the message is truncated.
-1	Insufficient memory available for message formatting services to function. The requested message is not returned.
-2	The SQLSTATE is in the wrong format. It must be alphanumeric and be either 2 or 5 digits in length.
-3	Message file inaccessible or incorrect.
-4	Line width is less than zero.
-5	Invalid *sqlca*, bad buffer address, or bad buffer length.

If the return code is -1 or -3, the message buffer will contain further information about the problem.

Related reference:
- "sqlaintp - Get Error Message" on page 257

Related samples:
- "checkerr.cbl -- Checks for and prints to the screen SQL warnings and errors (IBM COBOL)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

sqluadau - Get Authorizations

Reports the authorities of the current user from values found in the database manager configuration file and the authorization system catalog view (SYSCAT.DBAUTH).

Authorization:

None

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluadau */
/* ... */
SQL_API_RC SQL_API_FN
  sqluadau (
    struct sql_authorizations *pAuthorizations,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgadau */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgadau (
    struct sql_authorizations *pAuthorizations,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

pAuthorizations
Input/Output. Pointer to the *sql_authorizations* structure. This array of short integers indicates which authorizations the current user holds. The first element in the structure, *sql_authorizations_len*, must be initialized to the size of the buffer being passed, prior to calling this API.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
GET AUTHORIZATIONS :value
```

REXX API parameters:

value A compound REXX host variable to which the authorization level is returned. In the following, XXX represents the host variable name. Values are `0` for no, and `1` for yes.

XXX.0	Number of elements in the variable (always 18)
XXX.1	Direct SYSADM authority
XXX.2	Direct DBADM authority
XXX.3	Direct CREATETAB authority
XXX.4	Direct BINDADD authority

XXX.5	Direct CONNECT authority
XXX.6	Indirect SYSADM authority
XXX.7	Indirect DBADM authority
XXX.8	Indirect CREATETAB authority
XXX.9	Indirect BINDADD authority
XXX.10	Indirect CONNECT authority
XXX.11	Direct SYSCTRL authority
XXX.12	Indirect SYSCTRL authority
XXX.13	Direct SYSMAINT authority
XXX.14	Indirect SYSMAINT authority
XXX.15	Direct CREATE_NOT_FENC authority
XXX.16	Indirect CREATE_NOT_FENC authority
XXX.17	Direct IMPLICIT_SCHEMA authority
XXX.18	Indirect IMPLICIT_SCHEMA authority.
XXX.19	Direct LOAD authority.
XXX.20	Indirect LOAD authority.

Usage notes:

Direct authorities are acquired by explicit commands that grant the authorities to a user ID. Indirect authorities are based on authorities acquired by the groups to which a user belongs.

Note: PUBLIC is a special group to which all users belong.

If there are no errors, each element of the *sql_authorizations* structure contains a `0` or a `1`. A value of `1` indicates that the user holds that authorization; `0` indicates that the user does not.

Related reference:

- "SQL-AUTHORIZATIONS" on page 465
- "SQLCA" on page 478

Related samples:

- "dbauth.sqb -- How to grant and display authorities on a database (IBM COBOL)"

- "dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)"
- "inauth.sqc -- How to display authorities at instance level (C)"
- "dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)"
- "inauth.sqC -- How to display authorities at instance level (C++)"

sqludrdt - Redistribute Database Partition Group

Redistributes data across the database partitions in a database partition group. The current data distribution, whether it is uniform or skewed, can be specified. The redistribution algorithm selects the partitions to be moved based on the current data distribution.

This API can only be called from the catalog partition. Use the LIST DATABASE DIRECTORY command to determine which database partition server is the catalog partition for each database.

Scope:

This API affects all database partitions in the database partition group.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *dbadm*

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqludrdt */
/* ... */
SQL_API_RC SQL_API_FN
  sqludrdt (
    char *pNodeGroupName,
    char *pTargetPMapFileName,
    char *pDataDistFileName,
    SQL_PDB_NODE_TYPE *pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE *pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgdrdt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrdt (
    unsigned short NodeGroupNameLen,
    unsigned short TargetPMapFileNameLen,
    unsigned short DataDistFileNameLen,
    char *pNodeGroupName,
    char *pTargetPMapFileName,
    char *pDataDistFileName,
    SQL_PDB_NODE_TYPE *pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE *pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

NodeGroupNameLen
The length of the name of the database partition group.

TargetPMapFileNameLen
The length of the name of the target partitioning map file.

DataDistFileNameLen
The length of the name of the data distribution file.

pNodeGroupName
The name of the database partition group to be redistributed.

pTargetPMapFileName
The name of the file that contains the target partitioning map. If a directory path is not specified as part of the file name, the current directory is used. This parameter is used when the *DataRedistOption* value is T. The file should be in character format and contain either 4 096 entries (for a multiple-partition database partition group) or 1 entry (for a single-partition database partition group). Entries in the file indicate node numbers. Entries can be in free format.

pDataDistFileName
The name of the file that contains input distribution information. If a directory path is not specified as part of the file name, the current directory is used. This parameter is used when the *DataRedistOption*

value is U. The file should be in character format and contain 4 096 positive integer entries. Each entry in the file should indicate the weight of the corresponding partition. The sum of the 4 096 values should be less than or equal to 4 294 967 295.

pAddList
The list of database partitions to add to the database partition group during the data redistribution. Entries in the list must be in the form: SQL_PDB_NODE_TYPE.

AddCount
The number of database partitions to add to the database partition group.

pDropList
The list of database partitions to drop from the database partition group during the data redistribution. Entries in the list must be in the form: SQL_PDB_NODE_TYPE.

DropCount
The number of database partitions to drop from the database partition group.

DataRedistOption
A single character that indicates the type of data redistribution to be done. Possible values are:

U Specifies to redistribute the database partition group to achieve a balanced distribution. If *pDataDistFileName* is null, the current data distribution is assumed to be uniform (that is, each hash partition represents the same amount of data). If *pDataDistFileName* is not null, the values in this file are assumed to represent the current data distribution. When the *DataRedistOption* is U, the *pTargetPMapFileName* should be null.

Database partitions specified in the add list are added, and database partitions specified in the drop list are dropped from the database partition group.

T Specifies to redistribute the database partition group using *pTargetPMapFileName*. For this option, *pDataDistFileName*, *pAddList*, and *pDropList* should be null, and both *AddCount* and *DropCount* must be zero.

C Specifies to continue a redistribution operation that failed. For this option, *pTargetPMapFileName*, *pDataDistFileName*, *pAddList*, and *pDropList* should be null, and both *AddCount* and *DropCount* must be zero.

R Specifies to roll back a redistribution operation that failed. For this option, *pTargetPMapFileName*, *pDataDistFileName*, *pAddList*, and *pDropList* should be null, and both *AddCount* and *DropCount* must be zero.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Usage notes:

When a redistribution operation is done, a message file is written to:
- The `$HOME/sqllib/redist` directory on UNIX based systems, using the following format for subdirectories and file name: *database-name.nodegroup-name.timestamp*.
- The `$HOME\sqllib\redist\` directory on the Windows operating system, using the following format for subdirectories and file name: *database-name\first-eight-characters-of-the-nodegroup-name\date\time*.

The time stamp value is the time at which the API was called.

This utility performs intermittent COMMITs during processing.

Use the ALTER DATABASE PARTITION GROUP statement to add database partitions to a database partition group. This statement permits one to define the containers for the table spaces associated with the database partition group.

All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.

It is also recommended to update statistics by issuing the db2Runstats API after the redistribute database partition group operation has completed.

Database partition groups containing replicated summary tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.

Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables in the database partition group.

Related tasks:
- "Registering SQLEXEC, SQLDBS and SQLDB2 in REXX" in the *Application Development Guide: Programming Client Applications*

Related reference:
- "ALTER DATABASE PARTITION GROUP statement" in the *SQL Reference, Volume 2*
- "sqlarbnd - Rebind" on page 262
- "SQLCA" on page 478
- "LIST DATABASE DIRECTORY Command" in the *Command Reference*
- "REDISTRIBUTE DATABASE PARTITION GROUP Command" in the *Command Reference*
- "db2Runstats - Runstats" on page 228

sqluexpr - Export

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

or CONTROL or SELECT privilege on each participating table or view.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluexpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqluexpr (
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    sqlu_media_list *pLobFileList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluexpt_out *pOutputInfo,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgexpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgexpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    sqlu_media_list *pLobFileList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluexpt_out *pOutputInfo,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameter:

DataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the data file name.

FileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the file type.

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

pDataFileName
Input. A string containing the path and the name of the external file into which the data is to be exported.

pLobPathList
Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files are to be stored.

When file space is exhausted on the first path in this list, the API will use the second path, and so on.

pLobFileList
Input. An *sqlu_media_list* using *media_type* `SQLU_CLIENT_LOCATION`, and the *sqlu_location_entry* structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *pLobFilePath*), and then appending a 3-digit sequence number. For example, if the current LOB path is the directory `/u/foo/lob/path`, and the current LOB file name is `bar`, the created LOB files will be `/u/foo/lob/path/bar.001`, `/u/foo/lob/pah/bar.002`, and so on.

pDataDescriptor
Input. Pointer to an *sqldcol* structure specifying the column names for the output file. The value of the *dcolmeth* field determines how the remainder of the information provided in this parameter is interpreted by the export utility. Valid values for this parameter (defined in `sqlutil`) are:

SQL_METH_N
Names. Specify column names to be used in the output file.

SQL_METH_D
Default. Existing column names from the table are to be used in the output file. In this case, the number of columns and the column specification array are both ignored. The column names are derived from the output of the SELECT statement specified in *pActionString*.

pActionString
Input. Pointer to an *sqlchar* structure containing a valid dynamic SQL SELECT statement. The structure contains a 2-byte long field, followed by the characters that make up the SELECT statement. The SELECT statement specifies the data to be extracted from the database and written to the external file.

The columns for the external file (from *pDataDescriptor*), and the database columns from the SELECT statement, are matched according to their respective list/structure positions. The first column of data selected from the database is placed in the first column of the external file, and its column name is taken from the first element of the external column array.

pFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in `sqlutil`) are:

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

SQL_IXF
PC version of the Integrated Exchange Format, the preferred method for exporting data from a table. Data exported to this file format can later be imported or loaded into the same table or into another database manager table.

pFileTypeMod
Input. A pointer to an *sqldcol* structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, it is overwritten. If it does not exist, a file is created.

CallerAction
Input. An action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested export operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

pOutputInfo
Output. Returns the number of records exported to the target file.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
EXPORT :stmt TO datafile OF filetype
[MODIFIED BY :filetmod] [USING :dcoldata]
MESSAGES msgfile [ROWS EXPORTED :number]

CONTINUE EXPORT

STOP EXPORT
```

REXX API parameters:

stmt A REXX host variable containing a valid dynamic SQL SELECT statement. The statement specifies the data to be extracted from the database.

datafile
Name of the file into which the data is to be exported.

filetype
The format of the data in the export file. The supported file formats are:

DEL Delimited ASCII

WSF Worksheet format

IXF PC version of Integrated Exchange Format.

filetmod
A host variable containing additional processing options.

dcoldata
A compound REXX host variable containing the column names to be used in the export file. In the following, XXX represents the name of the host variable:

XXX.0	Number of columns (number of elements in the remainder of the variable).
XXX.1	First column name.
XXX.2	Second column name.
XXX.3	and so on.

If this parameter is NULL, or a value for *dcoldata* has not been specified, the utility uses the column names from the database table.

msgfile
File, path, or device name where error and warning messages are to be sent.

number
A host variable that will contain the number of exported rows.

Usage notes:

Be sure to complete all table operations and release all locks before starting an export operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

Table aliases can be used in the SELECT statement.

The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.

The export utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.

A warning message is issued if the number of columns (*dcolnum*) in the external column name array, *pDataDescriptor*, is not equal to the number of columns generated by the SELECT statement. In this case, the number of columns written to the external file is the lesser of the two numbers. Excess database columns or external column names are not used to generate the output file.

If the `db2uexpm.bnd` module or any other shipped `.bnd` files are bound manually, the **format** option on the binder must not be used.

PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

DB2 Connect can be used to export tables from DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF export is supported.

The export utility will not create multiple-part PC/IXF files when invoked from an AIX system.

Index definitions for a table are included in the PC/IXF file when the contents of a single database table are exported to a PC/IXF file with a *pActionString* beginning with `SELECT * FROM tablename`, and the *pDataDescriptor* parameter specifying default names. Indexes are not saved for views, or if the SELECT clause of the *pActionString* includes a join. A WHERE clause, a GROUP BY clause, or a HAVING clause in the *pActionString* will not prevent the saving of indexes. In all of these cases, when exporting from typed tables, the entire hierarchy must be exported.

The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form `SELECT * FROM tablename`.

When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and *select-statement* cannot be specified when exporting a hierarchy.

For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.

Note: Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.

DB2 Data Links Manager Considerations

To ensure that a consistent copy of the table and the corresponding files referenced by the DATALINK columns are copied for export, do the following:

1. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename SHARE.

   This ensures that no update transactions are in progress when EXPORT is run.
2. Issue the EXPORT command.
3. Run the **dlfm_export** utility at each Data Links server. Input to the **dlfm_export** utility is the control file name, which is generated by the export utility. This produces a tar (or equivalent) archive of the files listed within the control file. For Distributed File Systems (DFS), the **dlfm_export** utility will get the DCE network root credentials before archiving the files listed in the control file. **dlfm_export** does not capture the ACLs information of the files that are archived.
4. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename RESET.

   This makes the table available for updates.

EXPORT is executed as an SQL application. The rows and columns satisfying the SELECT statement conditions are extracted from the database. For the DATALINK columns, the SELECT statement should not specify any scalar function.

Successful execution of EXPORT results in generation of the following files:

- An export data file as specified in the EXPORT command. A DATALINK column value in this file has the same format as that used by the IMPORT and LOAD utilities. When the DATALINK column value is the SQL NULL value, handling is the same as that for other data types.
- Control files *server_name*, which are generated for each Data Links server. On the Windows NT operating system, a single control file, `ctrlfile.lst`, is used by all Data Links servers. For DFS, there is one control file for each cell. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows NT operating system, `ctrlfile.lst` is placed in the directory <data-file path>\dlfm\YYYYMMDD\HHMMSS). YYYYMMDD represents the date (year month day), and HHMMSS represents the time (hour minute second).

The **dlfm_export** utility is provided to export files from a Data Links server. This utility generates an archive file, which can be used to restore files in the target Data Links server.

Table 8. Valid File Type Modifiers (Export)

Modifier	**Description**
	All File Formats
lobsinfile	*lob-path* specifies the path to the files containing LOB values.

Table 8. Valid File Type Modifiers (Export) (continued)

Modifier	Description
	DEL (Delimited ASCII) File Format
chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[a] The single quotation mark (') can also be specified as a character string delimiter as follows: `modified by chardel''`
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data to this code page from the application code page during the export operation. For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. **Note:** The CODEPAGE modifier cannot be used with the LOBSINFILE modifier.

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[a]
	
	In the following example, `coldel;` causes the export utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 "export to temp of del modified by coldel;`
	`   select * from staff where dept = 20"`

datesiso	Date format. Causes all date data values to be exported in ISO format (*"YYYY-MM-DD"*).[b]
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[a]
dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [a] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.

Table 8. Valid File Type Modifiers (Export) (continued)

Modifier	Description
nodoubledel	Suppresses recognition of double character delimiters.[a]
WSF File Format	
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[c] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[c]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[c]
4	Creates a WSF file containing DBCS characters.

Notes:

1. The export utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the export operation fails, and an error code is returned.
2. [a] 417 lists restrictions that apply to the characters that can be used as delimiter overrides.
3. [b] The export utility normally writes
   - date data in *YYYYMMDD* format
   - char(date) data in "*YYYY-MM-DD*" format
   - time data in "*HH.MM.SS*" format
   - time stamp data in "*YYYY-MM-DD-HH.MM.SS.uuuuuu*" format

   Data contained in any datetime columns specified in the SELECT statement for the export operation will also be in these formats.
4. [c] These files can also be directed to a specific product by specifying an `L` for Lotus 1-2-3, or an `S` for Symphony in the *filetype-mod* parameter string. Only one value or product designator may be specified.

Delimiter restrictions:

It is the user's responsibility to ensure that the chosen delimiter character is not part of the data to be moved. If it is, unexpected errors may occur. The following restrictions apply to column, string, DATALINK, and decimal point delimiters when moving data:

- Delimiters are mutually exclusive.
- A delimiter cannot be binary zero, a line-feed character, a carriage-return, or a blank space.
- The default decimal point (.) cannot be a string delimiter.
- The following characters are specified differently by an ASCII-family code page and an EBCDIC-family code page:

- The Shift-In (0x0F) and the Shift-Out (0x0E) character cannot be delimiters for an EBCDIC MBCS data file.
- Delimiters for MBCS, EUC, or DBCS code pages cannot be greater than 0x40, except the default decimal point for EBCDIC MBCS data, which is 0x4b.
- Default delimiters for data files in ASCII code pages or EBCDIC MBCS code pages are:
  ```
  " (0x22, double quotation mark; string delimiter)
  , (0x2c, comma; column delimiter)
  ```
- Default delimiters for data files in EBCDIC SBCS code pages are:
  ```
  " (0x7F, double quotation mark; string delimiter)
  , (0x6B, comma; column delimiter)
  ```
- The default decimal point for ASCII data files is 0x2e (period).
- The default decimal point for EBCDIC data files is 0x4B (period).
- If the code page of the server is different from the code page of the client, it is recommended that the hex representation of non-default delimiters be specified. For example,
  ```
  db2 load from ... modified by chardel0x0C coldelX1e ...
  ```

The following information about support for double character delimiter recognition in DEL files applies to the export, import, and load utilities:

- Character delimiters are permitted within the character-based fields of a DEL file. This applies to fields of type CHAR, VARCHAR, LONG VARCHAR, or CLOB (except when `lobsinfile` is specified). Any pair of character delimiters found between the enclosing character delimiters is imported or loaded into the database. For example,
  ```
  "What a ""nice"" day!"
  ```
  will be imported as:
  ```
  What a "nice" day!
  ```
  In the case of export, the rule applies in reverse. For example,
  ```
  I am 6" tall.
  ```
  will be exported to a DEL file as:
  ```
  "I am 6"" tall."
  ```
- In a DBCS environment, the pipe (|) character delimiter is not supported.

Related concepts:

- "Moving DB2 Data Links Manager Using Export - Concepts" in the *Data Movement Utilities Guide and Reference*

Related reference:

- "SQLCA" on page 478
- "SQLCHAR" on page 479
- "SQLDCOL" on page 482
- "SQLUEXPT-OUT" on page 529
- "SQLU-MEDIA-LIST" on page 524

Related samples:
- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tload.sqb -- How to export and load table data (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

sqlugrpn - Get Row Partitioning Number

Returns the partition number and the database partition server number based on the partitioning key values. An application can use this information to determine at which database partition server a specific row of a table is stored.

The partitioning data structure, sqlupi, is the input for this API. The structure can be returned by the sqlugtpi API. Another input is the character representations of the corresponding partitioning key values. The output is a partition number generated by the partitioning strategy and the corresponding database partition server number from the partitioning map. If the partitioning map information is not provided, only the partition number is returned. This can be useful when analyzing data distribution.

The database manager does not need to be running when this API is called.

Scope:

This API can be invoked from any database partition server in the `db2nodes.cfg` file.

Authorization:

None

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlugrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugrpn (
    unsigned short num_ptrs,
    unsigned char **ptr_array,
    unsigned short *ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi *part_info,
    short *part_num,
    SQL_PDB_NODE_TYPE *node_num,
    unsigned short chklvl,
    struct sqlca *sqlca,
    short dataformat,
    void *pReserved1,
    void *pReserved2);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlggrpn */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggrpn (
    unsigned short num_ptrs,
    unsigned char **ptr_array,
    unsigned short *ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi *part_info,
    short *part_num,
    SQL_PDB_NODE_TYPE *node_num,
    unsigned short chklvl,
    struct sqlca *sqlca,
    short dataformat,
    void *pReserved1,
    void *pReserved2);
/* ... */
```

API parameters:

num_ptrs
: The number of pointers in *ptr_array*. The value must be the same as the one specified for *part_info*; that is, *part_info->sqld*.

ptr_array
: An array of pointers that points to the character representations of the corresponding values of each part of the partitioning key specified in *part_info*. If a null value is required, the corresponding pointer is set to null. For generated columns, this function does not generate values

for the row. The user is responsible for providing a value that will lead to the correct partitioning of the row.

ptr_lens
An array of unsigned integers that contains the lengths of the character representations of the corresponding values of each part of the partitioning key specified in *part_info*.

ctrycode
The country/region code of the target database.

This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

codepage
The code page of the target database.

This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

part_info
A pointer to the *sqlupi* structure.

part_num
A pointer to a 2-byte signed integer that is used to store the partition number.

node_num
A pointer to an SQL_PDB_NODE_TYPE field used to store the node number. If the pointer is null, no node number is returned.

chklvl An unsigned integer that specifies the level of checking that is done on input parameters. If the value specified is zero, no checking is done. If any non-zero value is specified, all input parameters are checked.

sqlca Output. A pointer to the *sqlca* structure.

dataformat
Specifies the representation of partitioning key values. Valid values are:

SQL_CHARSTRING_FORMAT
All partitioning key values are represented by character strings. This is the default value.

SQL_IMPLIEDDECIMAL_FORMAT
The location of an implied decimal point is determined by the column definition. For example, if the column definition is DECIMAL(8,2), the value 12345 is processed as 123.45.

SQL_PACKEDDECIMAL_FORMAT
All decimal column partitioning key values are in packed decimal format.

SQL_BINARYNUMERICS_FORMAT
All numeric partitioning key values are in big-endian binary format.

pReserved1
Reserved for future use.

pReserved2
Reserved for future use.

Usage notes:

Data types supported on the operating system are the same as those that can be defined as a partitioning key.

CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC must be converted to the target code page before this API is called.

For numeric and datetime data types, the character representations must be at the code page of the respective system where the API is invoked.

If *node_num* is not NULL, the partitioning map must be supplied; that is, `part_info->pmaplen` is either 2 or 8 192. Otherwise, SQLCODE -6038 is returned.

The partitioning key must be defined; that is, `part_info->sqld` must be greater than zero. Otherwise, SQLCODE -2032 is returned.

If a null value is assigned to a non-nullable partitioning column, SQLCODE -6039 is returned.

All the leading blanks and trailing blanks of the input character string are stripped, except for the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types, where only trailing blanks are stripped.

Related reference:
- "sqlugtpi - Get Table Partitioning Information" on page 423
- "sqludrdt - Redistribute Database Partition Group" on page 404
- "SQLCA" on page 478
- "SQLUPI" on page 532
- "GET DATABASE CONFIGURATION Command" in the *Command Reference*

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "db2CfgGet - Get Configuration Parameters" on page 39

sqlugtpi - Get Table Partitioning Information

Allows an application to obtain the partitioning information for a table. The partitioning information includes the partitioning map and the column definitions of the partitioning key. Information returned by this API can be passed to the sqlugrpn API to determine the partition number and the database partition server number for any row in the table.

To use this API, the application must be connected to the database that contains the table for which partitioning information is being requested.

Scope:

This API can be executed on any database partition server defined in the `db2nodes.cfg` file.

Authorization:

For the table being referenced, a user must have at least one of the following:
- *sysadm* authority
- *dbadm* authority
- CONTROL privilege
- SELECT privilege

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlugtpi */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugtpi (
    unsigned char *tablename,
    struct sqlupi *part_info,
    struct sqlca *sqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlggtpi */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtpi (
    unsigned short tn_length,
    unsigned char *tablename,
    struct sqlupi *part_info,
    struct sqlca *sqlca);
/* ... */
```

API parameters:

tn_length
A 2-byte unsigned integer with the length of the table name.

tablename
The fully qualified name of the table.

part_info
A pointer to the *sqlupi* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "sqlugrpn - Get Row Partitioning Number" on page 419
- "sqludrdt - Redistribute Database Partition Group" on page 404
- "SQLCA" on page 478
- "SQLUPI" on page 532

sqluimpr - Import

Inserts data from an external file with a supported file format into a table, hierarchy, or view. A faster alternative is Load however, the load utility does not support loading data at the hierarchy level.

Authorization:

- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*

  - CONTROL privilege on each participating table or view
  - INSERT and SELECT privilege on each participating table or view.
- IMPORT to an existing table using the INSERT_UPDATE, REPLACE, or the REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on the table or view.
- IMPORT to a table or a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database, and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists.
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used.
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on every sub-table in the hierarchy.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

sqluimpr - Import

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluimpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqluimpr (
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluimpt_in *pImportInfoIn,
    struct sqluimpt_out *pImportInfoOut,
    sqlint32 *pNullIndicators,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqluimpr */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgimpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char *pDataFileName,
    sqlu_media_list *pLobPathList,
    struct sqldcol *pDataDescriptor,
    struct sqlchar *pActionString,
    char *pFileType,
    struct sqlchar *pFileTypeMod,
    char *pMsgFileName,
    short CallerAction,
    struct sqluimpt_in *pImportInfoIn,
    struct sqluimpt_out *pImportInfoOut,
    sqlint32 *NullIndicators,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

DataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the input file name.

FileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the input file type.

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

pDataFileName
Input. A string containing the path and the name of the external input file from which the data is to be imported.

pLobPathList
Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files can be found.

pDataDescriptor
Input. Pointer to an *sqldcol* structure containing information about the columns being selected for import from the external file. The value of the *dcolmeth* field determines how the remainder of the information provided in this parameter is interpreted by the import utility. Valid values for this parameter (defined in `sqlutil`) are:

SQL_METH_N
Names. Selection of columns from the external input file is by column name.

SQL_METH_P
Positions. Selection of columns from the external input file is by column position.

SQL_METH_L
Locations. Selection of columns from the external input file is by column location. The database manager rejects an import call with a location pair that is invalid because of any one of the following conditions:

- Either the beginning or the ending location is not in the range from 1 to the largest signed 2-byte integer.
- The ending location is smaller than the beginning location.
- The input column width defined by the location pair is not compatible with the type and the length of the target column.

A location pair with both locations equal to zero indicates that a nullable column is to be filled with NULLs.

SQL_METH_D
Default. If *pDataDescriptor* is NULL, or is set to `SQL_METH_D`, default selection of columns from the external input file is done. In this case, the number of columns and the column specification array are both ignored. For DEL, IXF, or WSF files, the first *n* columns of data in the external input file are taken in their natural order, where *n* is the number of database columns into which the data is to be imported. For ASC files the selection of columns from the external input files is to be read in from the file specified by the *pFileTypeMod* modified POSITIONSFILE.

pActionString
Input. Pointer to an *sqlchar* structure containing a 2-byte long field, followed by an array of characters identifying the columns into which data is to be imported.

The character array is of the form:

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
[DATALINK SPECIFICATION datalink-spec]
```

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds the imported rows if their primary key values are not in the table, and uses them for update if their primary key values are found. This option is only valid if the target table has a primary key, and the specified (or implied) list of target columns being imported includes all columns for the primary key. This option cannot be applied to views.

REPLACE
Deletes all existing data from the table by truncating the table object, and inserts the imported data. The table definition and the index definitions are not changed. (Indexes are deleted and replaced if `indexixf` is in *FileTypeMod*, and *FileType* is `SQL_IXF`.) If the table is not already defined, an error is returned.

Attention: If an error occurs after the existing data is deleted, that data is lost.

CREATE
Creates the table definition and the row contents using the information in the specified PC/IXF file, if the specified table is not defined. If the file was previously exported by DB2, indexes are also created. If the specified table is already defined, an error is returned. This option is valid for the PC/IXF file format only.

REPLACE_CREATE
Replaces the table contents using the PC/IXF row information in the PC/IXF file, if the specified table is defined. If the table is not already defined, the table definition and row contents are created using the information in the specified PC/IXF file. If the PC/IXF file was previously exported by DB2, indexes are also created. This option is valid for the PC/IXF file format only.

Attention: If an error occurs after the existing data is deleted, that data is lost.

tname The name of the table, typed table, view, or object view into which the data is to be inserted. An alias for REPLACE, INSERT_UPDATE, or INSERT can be specified, except in the case of a down-level server, when a qualified or unqualified name should be specified. If it is a view, it cannot be a read-only view.

tcolumn-list
A list of table or view column names into which the data is to be inserted. The column names must be separated by commas. If column names are not specified, column names as defined in the CREATE TABLE or the ALTER TABLE statement are used. If no column list is specified for typed tables, data is inserted into all columns within each sub-table.

sub-table-name
Specifies a parent table when creating one or more sub-tables under the CREATE option.

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the *traversal-order-list*.

HIERARCHY
Specifies that hierarchical data is to be imported.

STARTING
Keyword for hierarchy only. Specifies that the default order, starting from a given sub-table name, is to be used.

UNDER
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created under a given sub-table.

AS ROOT TABLE
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created as a stand-alone hierarchy.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the IMPORT command.

The *tname* and the *tcolumn-list* parameters correspond to the *tablename* and the *colname* lists of SQL INSERT statements, and have the same restrictions.

The columns in *tcolumn-list* and the external columns (either specified or implied) are matched according to their position in the list or the structure (data from the first column specified in the *sqldcol* structure is inserted into the table or view field corresponding to the first element of the *tcolumn-list*).

If unequal numbers of columns are specified, the number of columns actually processed is the lesser of the two numbers. This could result in an error (because there are no values to place in some non-nullable table fields) or an informational message (because some external file columns are ignored).

pFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in `sqlutil`) are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integrated Exchange Format, the preferred

method for exporting data from a table so that it can be imported later into the same table or into another database manager table.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

pFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, it is appended to. If it does not exist, a file is created.

CallerAction
Input. An action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested import operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action

requested by the utility was not performed, and the utility is to terminate processing the initial request.

pImportInfoIn
Input. Optional pointer to the *sqluimpt_in* structure containing additional input parameters.

pImportInfoOut
Output. Optional pointer to the *sqluimpt_out* structure containing additional output parameters.

NullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. The number of elements in this array must match the number of columns in the input file; there is a one-to-one ordered correspondence between the elements of this array and the columns being imported from the data file. Therefore, the number of elements must equal the *dcolnum* field of the *pDataDescriptor* parameter. Each element of the array contains a number identifying a column in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified column in the data file must contain a `Y` or an `N`. A `Y` indicates that the table column data is NULL, and `N` indicates that the table column data is not NULL.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
IMPORT FROM datafile OF filetype
[MODIFIED BY :filetmod]
[METHOD {L|N|P} USING :dcoldata]
[COMMITCOUNT :commitcnt] [RESTARTCOUNT :restartcnt]
MESSAGES msgfile
{INSERT|REPLACE|CREATE|INSERT_UPDATE|REPLACE_CREATE}
INTO tname [(:columns)]
[OUTPUT INTO :output]

CONTINUE IMPORT

STOP IMPORT
```

REXX API parameter:

datafile
Name of the file from which the data is to be imported.

filetype
The format of the data in the external import file. The supported file formats are:

DEL	Delimited ASCII
ASC	Non-delimited ASCII
WSF	Worksheet format
IXF	PC version of Integrated Exchange Format.

filetmod
A host variable containing additional processing options.

L | N | P
A character specifying the method to be used to select columns within the external input file. Valid values are:

L	Location
N	Name
P	Position.

dcoldata
A compound REXX host variable containing information about the columns selected for import from the external input file. The content of the structure depends upon the specified *method*. In the following, XXX represents the name of the host variable:

- Location method

XXX.0	Number of elements in the remainder of the variable
XXX.1	A number representing the starting location of this column in the input file. This column becomes the first column in the database table.
XXX.2	A number representing the ending location of the column.
XXX.3	A number representing the starting location of this column in the input file. This column becomes the second column in the database table.
XXX.4	A number representing the ending location of the column.
XXX.5	and so on.

- Name method

XXX.0	Number of column names contained in the host variable.
XXX.1	First name.
XXX.2	Second name.

XXX.3 and so on.

- Position method

  XXX.0 Number of column positions contained in the host variable.

  XXX.1 A column position in the external input file.

  XXX.2 A column position in the external input file.

  XXX.3 and so on.

tname Name of the target table or view. Data cannot be imported to a read-only view.

columns
A REXX host variable containing the names of the columns in the table or the view into which the data is to be inserted. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns.

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

msgfile
File, path, or device name where error and warning messages are to be sent.

commitcnt
Performs a COMMIT after every *commitcnt* records are imported.

restartcnt
Specifies that an import operation is to be started at record *restartcnt* + 1. The first *restartcnt* records are skipped.

output
A compound REXX host variable into which information from the import operation is passed. In the following, XXX represents the name of the host variable:

XXX.1 Number of records read from the external input file during the import operation.

XXX.2 Number of records skipped before inserting or updating begins.

XXX.3 Number of rows inserted into the target table.

XXX.4 Number of rows in the target table updated with information from the imported records.

XXX.5 Number of records that could not be imported.

XXX.6 Number of records imported successfully and committed to the database, including rows inserted, updated, skipped, and rejected.

Usage notes:

Be sure to complete all table operations and release all locks before starting an import operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

The import utility adds rows to the target table using the SQL INSERT statement. The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the *restartcnt* parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the *commitcnt* parameter is not zero. A full log results in a ROLLBACK.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

One cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:
1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30KB of message information and the last 30KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes. Non-default values for *pDataDescriptor*, or specifying an explicit list of table columns in *pActionString*, makes importing to a remote database slower.

The database table or hierarchy must exist before data in the ASC, DEL, or WSF file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 for AIX clients.

For table objects on an 8KB page that are close to the limit of 1012 columns, import of PC/IXF data files may cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of DEL or ASC files.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The restartcnt parameter, but not the commitcnt parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file.

The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

On the Windows NT operating system:
- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

DB2 Data Links Manager Considerations

Before running the DB2 import utility, do the following:

1. Copy the files that will be referenced to the appropriate Data Links servers. The **dlfm_import** utility can be used to extract files from an archive that is generated by the **dlfm_export** utility.
2. Register the required prefix names to the DB2 Data Links Managers. There may be other administrative tasks, such as registering the database, if required.
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the original configuration's Data Links servers are the same at the target location, the Data Links server names need not be updated.) For Distributed File Systems (DFS), update the cell name information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required.
4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file. For DFS, define the cells at the target configuration in the DB2 Data Links Manager configuration file.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

During the insert operation, DATALINK column processing links the files in the appropriate Data Links servers according to the column specifications at the target database.

Representation of DATALINK Information in an Input File

Table 9. Valid File Type Modifiers (Import)

Modifier	Description
All File Formats	

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**
compound=*x*	*x* is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail. The transaction log must be large enough to accommodate either the number of rows specified by COMMITCOUNT, or the number of rows in the data file if COMMITCOUNT is not specified. It is therefore recommended that the COMMITCOUNT option be specified to avoid transaction log overflow. This modifier is incompatible with INSERT_UPDATE mode, hierarchical tables, and the following modifiers: `usedefaults`, `identitymissing`, `identityignore`, `generatedmissing`, and `generatedignore`.
generatedignore	This modifier informs the import utility that data for all generated columns is present in the data file but should be ignored. This results in all values for the generated columns being generated by the utility. This modifier cannot be used with the `generatedmissing` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated columns (not even NULLs), and will therefore generate a value for each row. This modifier cannot be used with the `generatedignore` modifier.
identityignore	This modifier informs the import utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with the `identitymissing` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with the `identityignore` modifier.
lobsinfile	*lob-path* specifies the path to the files containing LOB values.

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If a default value can be specified for a column, the default value is loaded • If the column is nullable, and a default value cannot be specified for that column, a NULL is loaded • If the column is not nullable, and a default value cannot be specified, an error is returned, and the utility stops processing.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: ",," is specified for the column • For ASC files: The NULL indicator is set to yes for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data to this code page from the application code page during the import operation. The following rules apply: • For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 ans x7F, inclusive. This refers to ASCII symbols and code points. **Notes:** 1. The CODEPAGE modifier cannot be used with the LOBSINFILE modifier. 2. If data expansion occurs when the code page is converted from the application code page to the database code page, the data may be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[a] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M    - Month (one or two digits ranging from 1 - 12)`
	`MM   - Month (two digits ranging from 1 - 12;`
	`         mutually exclusive with M)`
	`D    - Day (one or two digits ranging from 1 - 31)`
	`DD   - Day (two digits ranging from 1 - 31;`
	`         mutually exclusive with D)`
	`DDD  - Day of the year (three digits ranging`
	`         from 001 - 366; mutually exclusive`
	`         with other day or month elements)`
	
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description

timeformat="*x*"	*x* is the format of the time in the source file.[a] Valid time elements are:
	`H     - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH    - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M     - Minute (one or two digits ranging from 0 - 59)`
	`MM    - Minute (two digits ranging from 0 - 59; mutually exclusive with M)`
	`S     - Second (one or two digits ranging from 0 - 59)`
	`SS    - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`TT    - Meridian indicator (AM or PM)`
	
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[a] Valid time stamp elements are:
	`YYYY   - Year (four digits ranging from 0000 - 9999)`
	`M      - Month (one or two digits ranging from 1 - 12)`
	`MM     - Month (two digits ranging from 1 - 12; mutually exclusive with M, month)`
	`D      - Day (one or two digits ranging from 1 - 31)`
	`DD     - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD    - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	`H      - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH     - Hour (two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system; mutually exclusive with H)`
	`M      - Minute (one or two digits ranging from 0 - 59)`
	`MM     - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)`
	`S      - Second (one or two digits ranging from 0 - 59)`
	`SS     - Second (two digits ranging from 0 - 59; mutually exclusive with S)`
	`SSSSS  - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999)`
	`TT     - Meridian indicator (AM or PM)`
	
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`   modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`   insert into schedule`

ASC (Non-delimited ASCII) File Format	

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[b] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the null indicator character is specified to be the letter `N`, then `n` is also recognized as a null indicator.
reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.

striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.
	
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	
	`db2 import from myfile.asc of asc`
	`   modified by striptblanks`
	`   method l (1 10, 12 15) messages msgs.txt`
	`   insert into staff`
	
	This option cannot be specified together with `striptnulls`. These are mutually exclusive options.
	Note: This option replaces the obsolete `t` option, which is supported for back-level compatibility only.

striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.
DEL (Delimited ASCII) File Format	

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[bc]
	
	The single quotation mark (') can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (') it encounters as a character string delimiter:
	
	`db2 "import from myfile.del of del`
	`   modified by chardel''`
	`   method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[bc]
	
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	
	`db2 import from myfile.del of del`
	`   modified by coldel;`
	`   messages msgs.txt insert into staff`

datesiso	Date format. Causes all date data values to be imported in ISO format.
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.

decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[bc]
	
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	
	`db2 "import from myfile.del of del`
	`   modified by chardel'`
	`   decpt; messages msgs.txt insert into staff"`

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 import ... modified by delprioritychar ...`
	
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [bc] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.
keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
nodoubledel	Suppresses recognition of double character delimiters. For more information, see the "Delimiter restrictions" described in the sqluexpr API.
IXF File Format	
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to import each row.

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Table 9. Valid File Type Modifiers (Import) (continued)

Modifier	Description

Notes:

1. The import utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the import operation fails, and an error code is returned.
2. [a] Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Note: Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
3. [b] The character must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
4. [c] The “Delimiter restrictions” described in the **sqluexpr** API list restrictions that apply to the characters that can be used as delimiter overrides.

Related reference:
- "SQLCA" on page 478
- "SQLU-MEDIA-LIST" on page 524

Related samples:
- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbmove.sqc -- How to move table data (C)"
- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tbmove.sqC -- How to move table data (C++)"

sqlurcon - Reconcile

Validates the references to files for the DATALINK data of a table. The rows for which the references to files cannot be established are copied to the exception table (if specified), and modified in the input table.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table.

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqlurcon */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurcon (
    char *pTableName,
    char *pExTableName,
    char *pReportFileName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgrcon */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrcon (
    unsigned short TableNameLen,
    char *pTableName,
    unsigned short ExTableNameLen,
    char *pExTableName,
    unsigned short ReportFleNameLen,
    char *pReportFileName,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

TableNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the table name.

pTableName
: Input. Specifies the table on which reconciliation is to be performed. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the current authorization ID.

ExTableNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the exception table name.

pExTableName
: Input. Specifies the exception table into which rows that encounter link failures for DATALINK values are to be copied.

ReportFileNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the report file name.

pReportFileName
: Input. Specifies the file that will contain information about the files that are unlinked during reconciliation. The name must be fully qualified (for example, `/u/johnh/report`). The reconcile utility appends a `.ulk` extension to the specified file name (for example, `report.ulk`).

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

During reconciliation, attempts are made to link files which exist according to table data, but which do not exist according to Data Links File Manager metadata, if no other conflict exists.

Reconciliation is performed with respect to all DATALINK data in the table. If file references cannot be re-established, the violating rows are inserted into the exception table (if specified). These rows are not deleted from the input table. To ensure file reference integrity, the offending DATALINK values are nulled. If the column is defined as not nullable, the DATALINK values are replaced by a zero length URL.

If an exception table is not specified, the DATALINK column values for which file references cannot be re-established are copied to an exception report file (`<`*`pReportFileName`*`>.exp`), along with the column ID and a comment.

At the end of the reconciliation process, the table is taken out of datalink reconcile pending (DRP) state.

Related reference:
- "SQLCA" on page 478

sqluvqdp - Quiesce Table Spaces for Table

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive. There are three possible table space states resulting from the quiesce function: QUIESCED SHARE, QUIESCED UPDATE, and QUIESCED EXCLUSIVE.

Scope:

In a single-partition database environment, this API quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load. In a partitioned database environment, this API acts locally on a database partition. It quiesces only that portion of table spaces belonging to the database partition on which the load is performed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

API include file:

sqlutil.h

C API syntax:

```
/* File: sqlutil.h */
/* API: sqluvqdp */
/* ... */
SQL_API_RC SQL_API_FN
  sqluvqdp (
    char *pTableName,
    sqlint32 QuiesceMode,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

Generic API syntax:

```
/* File: sqlutil.h */
/* API: sqlgvqdp */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgvqdp (
    unsigned short TableNameLen,
    char *pTableName,
    sqlint32 QuiesceMode,
    void *pReserved,
    struct sqlca *pSqlca);
/* ... */
```

API parameters:

TableNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the table name.

pTableName
: Input. A string containing the table name as used in the system catalog. This may be a two-part name with the *schema* and the table

name separated by a period (.). If the *schema* is not provided, the CURRENT SCHEMA will be used. The table cannot be a system catalog table. This field is mandatory.

QuiesceMode
Input. Specifies the quiesce mode. Valid values (defined in `sqlutil`) are:

SQLU_QUIESCEMODE_SHARE
For share mode

SQLU_QUIESCEMODE_INTENT_UPDATE
For intent to update mode

SQLU_QUIESCEMODE_EXCLUSIVE
For exclusive mode

SQLU_QUIESCEMODE_RESET
To reset the state of the table spaces to normal if either of the following is true:

- The caller owns the quiesce
- The caller who sets the quiesce disconnects, creating a ″phantom quiesce″

SQLU_QUIESCEMODE_RESET_OWNED
To reset the state of the table spaces to normal if the caller owns the quiesce.

This field is mandatory.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure.

REXX API syntax:

```
QUIESCE TABLESPACES FOR TABLE table_name
{SHARE | INTENT TO UPDATE | EXCLUSIVE | RESET}
```

REXX API parameters:

table_name
Name of the table as used in the system catalog. This may be a two-part name with the *schema* and the table name separated by a period (.). If the *schema* is not provided, the CURRENT SCHEMA will be used.

Usage notes:

This API is not supported for declared temporary tables.

When the quiesce share request is received, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces is recorded in the table space table, along with the authorization ID and the database agent ID of the quiescer, so that the state is persistent.

The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces will be allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

When the quiesce exclusive request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer), however, has exclusive access to the table and the table spaces.

When a quiesce update request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces with the quiescer is recorded in the table space table.

There is a limit of five quiescers on a table space at any given time. Since QUIESCED EXCLUSIVE is incompatible with any other state, and QUIESCED UPDATE is incompatible with another QUIESCED UPDATE, the five quiescer limit, if reached, must have at least four QUIESCED SHARE and at most one QUIESCED UPDATE.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

The quiesced state of a table space must be reset explicitly by using SQLU_QUIESCEMODE_RESET.

Related reference:

- "SQLCA" on page 478

- "db2DatabaseQuiesce - Database Quiesce" on page 51
- "db2InstanceQuiesce - Instance Quiesce" on page 101

Related samples:

- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"
- "tload.sqb -- How to export and load table data (IBM COBOL)"

Chapter 2. Additional REXX APIs

This section describes DB2 application programming interfaces that are only supported in the REXX programming language.

Change Isolation Level (REXX)

Changes the way that DB2 isolates data from other processes while a database is being accessed.

Authorization:

None

Required connection:

None

REXX API syntax:

```
CHANGE SQLISL TO {RR|CS|UR|RS|NC}
```

REXX API parameters:

RR	Repeatable read.
CS	Cursor stability. This is the default.
UR	Uncommitted read.
RS	Read stability.
NC	No commit.

Related reference:

- "REXX Samples" in the *Application Development Guide: Building and Running Applications*

Chapter 3. Data Structures

This section describes the data structures used to access the database manager.

db2HistData

This structure is used to return information after a call to db2HistoryGetEntry.

Table 10. Fields in the db2HistData Structure

Field Name	Data Type	Description
ioHistDataID	char(8)	An 8-byte structure identifier and “eye-catcher” for storage dumps. The only valid value is “SQLUHINF”. No symbolic definition for this string exists.
oObjectPart	db2Char	The first 14 characters are a time stamp with format *yyyymmddhhnnss*, indicating when the operation was begun. The next 3 characters are a sequence number. Each backup operation can result in multiple entries in this file when the backup image is saved in multiple files or on multiple tapes. The sequence number allows multiple locations to be specified. Restore and load operations have only a single entry in this file, which corresponds to sequence number ’001’ of the corresponding backup. The time stamp, combined with the sequence number, must be unique.
oEndTime	db2Char	A time stamp with format *yyyymmddhhnnss*, indicating when the operation was completed.
oFirstLog	db2Char	The earliest log file ID (ranging from S0000000 to S9999999): • Required to apply rollforward recovery for an online backup • Required to apply rollforward recovery for an offline backup • Applied after restoring a full database or table space level backup that was current when the load started.

Table 10. Fields in the db2HistData Structure (continued)

Field Name	Data Type	Description
oLastLog	db2Char	The latest log file ID (ranging from S0000000 to S9999999): • Required to apply rollforward recovery for an online backup • Required to apply rollforward recovery to the current point in time for an offline backup • Applied after restoring a full database or table space level backup that was current when the load operation finished (will be the same as *oFirstLog* if roll forward recovery is not applied).
oID	db2Char	Unique backup or table identifier.
oTableQualifier	db2Char	Table qualifier.
oTableName	db2Char	Table name.
oLocation	db2Char	For backups and load copies, this field indicates where the data has been saved. For operations that require multiple entries in the file, the sequence number defined by *oObjectPart* identifies which part of the backup is found in the specified location. For restore and load operations, the location always identifies where the first part of the data restored or loaded (corresponding to sequence '001' for multi-part backups) has been saved. The data in *oLocation* is interpreted differently, depending on *oDeviceType*: • For disk or diskette (`D` or `K`), a fully qualified file name • For tape (`T`), a volume label • For TSM (`A`), the server name • For user exit or other (`U` or `O`), free form text.
oComment	db2Char	Free form text comment.
oCommandText	db2Char	Command text, or DDL.
oLastLSN	SQLU_LSN	Last log sequence number.
oEID	Structure	Unique entry identifier.
poEventSQLCA	Structure	Result *sqlca* of the recorded event.
poTablespace	db2Char	A list of table space names.

Table 10. Fields in the db2HistData Structure (continued)

Field Name	Data Type	Description
ioNumTablespaces	db2Uint32	Number of entries in the *poTablespace* list. Each table space backup contains one or more table spaces. Each table space restore operation replaces one or more table spaces. If this field is not zero (indicating a table space level backup or restore), the next lines in this file contain the name of the table space backed up or restored, represented by an 18-character string. One table space name appears on each line.
oOperation	char	See Table 11.
oObject	char	Granularity of the operation: `D` for full database, `P` for table space, and `T` for table.
oOptype	char	See Table 12 on page 462.
oStatus	char	Entry status: `A` for action, `D` for deleted (future use), `E` for expired, `I` for inactive, `N` for not yet committed, `Y` for committed or active, `a` for active backup, but some datalink servers have not yet completed the backup, and `i` for inactive backup, but some datalink servers have not yet completed the backup.
oDeviceType	char	Device type. This field determines how the *oLocation* field is interpreted: `A` for TSM, `C` for client, `D` for disk, `K` for diskette, `L` for local, `O` for other (for other vendor device support), `P` for pipe, `Q` for cursor, `S` for server, `T` for tape, and `U` for user exit.

Table 11. Valid oOperation Values in the db2HistData Structure

Value	Description	C Definition	COBOL/FORTRAN Definition
A	add table space	DB2HISTORY_OP_ADD_ TABLESPACE	DB2HIST_OP_ADD_ TABLESPACE
B	backup	DB2HISTORY_OP_BACKUP	DB2HIST_OP_BACKUP
C	load copy	DB2HISTORY_OP_LOAD_COPY	DB2HIST_OP_LOAD_COPY
D	dropped table	DB2HISTORY_OP_DROPPED_ TABLE	DB2HIST_OP_DROPPED_TABLE
F	rollforward	DB2HISTORY_OP_ROLLFWD	DB2HIST_OP_ROLLFWD
G	reorganize table	DB2HISTORY_OP_REORG	DB2HIST_OP_REORG
L	load	DB2HISTORY_OP_LOAD	DB2HIST_OP_LOAD
N	rename table space	DB2HISTORY_OP_REN_ TABLESPACE	DB2HIST_OP_REN_ TABLESPACE
O	drop table space	DB2HISTORY_OP_DROP_ TABLESPACE	DB2HIST_OP_DROP_ TABLESPACE
Q	quiesce	DB2HISTORY_OP_QUIESCE	DB2HIST_OP_QUIESCE

Table 11. Valid oOperation Values in the db2HistData Structure (continued)

Value	Description	C Definition	COBOL/FORTRAN Definition
R	restore	DB2HISTORY_OP_RESTORE	DB2HIST_OP_RESTORE
T	alter table space	DB2HISTORY_OP_ALT_ TABLESPACE	DB2HIST_OP_ALT_TBS
U	unload	DB2HISTORY_OP_UNLOAD	DB2HIST_OP_UNLOAD

Table 12. Valid oOptype Values in the db2HistData Structure

oOperation	oOptype	Description	C/COBOL/FORTRAN Definition
B	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
	I	incremental offline	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	incremental online	DB2HISTORY_OPTYPE_INCR_ ONLINE
	D	delta offline	DB2HISTORY_OPTYPE_DELTA_ OFFLINE
	E	delta online	DB2HISTORY_OPTYPE_DELTA_ ONLINE
F	E	end of logs	DB2HISTORY_OPTYPE_EOL
	P	point in time	DB2HISTORY_OPTYPE_PIT
G	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
L	I	insert	DB2HISTORY_OPTYPE_INSERT
	R	replace	DB2HISTORY_OPTYPE_REPLACE
Q	S	quiesce share	DB2HISTORY_OPTYPE_SHARE
	U	quiesce update	DB2HISTORY_OPTYPE_UPDATE
	X	quiesce exclusive	DB2HISTORY_OPTYPE_EXCL
	Z	quiesce reset	DB2HISTORY_OPTYPE_RESET
R	F	offline	DB2HISTORY_OPTYPE_OFFLINE
	N	online	DB2HISTORY_OPTYPE_ONLINE
	I	incremental offline	DB2HISTORY_OPTYPE_INCR_ OFFLINE
	O	incremental online	DB2HISTORY_OPTYPE_INCR_ ONLINE
T	C	add containers	DB2HISTORY_OPTYPE_ADD_CONT
	R	rebalance	DB2HISTORY_OPTYPE_REB

Table 13. Fields in the db2Char Structure

Field Name	Data Type	Description
pioData	char	A pointer to a character data buffer. If NULL, no data will be returned.
iLength	db2Uint32	Input. The size of the *pioData* buffer.
oLength	db2Uint32	Output. The number of valid characters of data in the *pioData* buffer.

Table 14. Fields in the db2HistoryEID Structure

Field Name	Data Type	Description
ioNode	SQL_PDB_NODE_TYPE	Node number.
ioHID	db2Uint32	Local history file entry ID.

Language syntax:

C Structure

```
/* File: db2ApiDf.h */
/* ... */
typedef SQL_STRUCTURE db2HistoryData
{
  char ioHistDataID[8];
  db2Char oObjectPart;
  db2Char oEndTime;
  db2Char oFirstLog;
  db2Char oLastLog;
  db2Char oID;
  db2Char oTableQualifier;
  db2Char oTableName;
  db2Char oLocation;
  db2Char oComment;
  db2Char oCommandText;
  SQLU_LSN oLastLSN;
  db2HistoryEID oEID;
  struct sqlca * poEventSQLCA;
  db2Char * poTablespace;
  db2Uint32 ioNumTablespaces;
  char oOperation;
  char oObject;
  char oOptype;
  char oStatus;
  char oDeviceType
} db2HistoryData;

typedef SQL_STRUCTURE db2Char
{
  char * pioData;
  db2Uint32 ioLength
} db2Char;

typedef SQL_STRUCTURE db2HistoryEID
{
  SQL_PDB_NODE_TYPE ioNode;
  db2Uint32 ioHID
} db2HistoryEID;
/* ... */
```

Related reference:

- “db2HistoryGetEntry - Get Next History File Entry” on page 82
- “SQLCA” on page 478

SQL-AUTHORIZATIONS

This structure is used to return information after a call to the sqluadau API. The data type of all fields is SMALLINT. The first half of the following table contains authorities granted directly to a user. The second half of the table contains authorities granted to the groups to which a user belongs.

Table 15. Fields in the SQL-AUTHORIZATIONS Structure

Field Name	Description
SQL_AUTHORIZATIONS_LEN	Size of structure.
SQL_SYSADM_AUTH	SYSADM authority.
SQL_SYSCTRL_AUTH	SYSCTRL authority.
SQL_SYSMAINT_AUTH	SYSMAINT authority.
SQL_DBADM_AUTH	DBADM authority.
SQL_CREATETAB_AUTH	CREATETAB authority.
SQL_CREATET_NOT_FENC_AUTH	CREATE_NOT_FENCED authority.
SQL_BINDADD_AUTH	BINDADD authority.
SQL_CONNECT_AUTH	CONNECT authority.
SQL_IMPLICIT_SCHEMA_AUTH	IMPLICIT_SCHEMA authority.
SQL_LOAD_AUTH	LOAD authority.
SQL_SYSADM_GRP_AUTH	User belongs to a group which holds SYSADM authority.
SQL_SYSCTRL_GRP_AUTH	User belongs to a group which holds SYSCTRL authority.
SQL_SYSMAINT_GRP_AUTH	User belongs to a group which holds SYSMAINT authority.
SQL_DBADM_GRP_AUTH	User belongs to a group which holds DBADM authority.
SQL_CREATETAB_GRP_AUTH	User belongs to a group which holds CREATETAB authority.
SQL_CREATE_NON_FENC_GRP_AUTH	User belongs to a group which holds CREATE_NOT_FENCED authority.
SQL_BINDADD_GRP_AUTH	User belongs to a group which holds BINDADD authority.
SQL_CONNECT_GRP_AUTH	User belongs to a group which holds CONNECT authority.
SQL_IMPLICIT_SCHEMA_GRP_AUTH	User belongs to a group which holds IMPLICIT_SCHEMA authority.
SQL_LOAD_GRP_AUTH	User belongs to a group which holds LOAD authority.
Note: SYSADM, SYSMAINT, and SYSCTRL are only indirect authorities and cannot be granted directly to the user. They are available only through the groups to which the user belongs.	

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-AUTHORIZATIONS */
/* ... */
SQL_STRUCTURE sql_authorizations
{
  short           sql_authorizations_len;
  short           sql_sysadm_auth;
  short           sql_dbadm_auth;
  short           sql_createtab_auth;
  short           sql_bindadd_auth;
  short           sql_connect_auth;
  short           sql_sysadm_grp_auth;
  short           sql_dbadm_grp_auth;
  short           sql_createtab_grp_auth;
  short           sql_bindadd_grp_auth;
  short           sql_connect_grp_auth;
  short           sql_sysctrl_auth;
  short           sql_sysctrl_grp_auth;
  short           sql_sysmaint_auth;
  short           sql_sysmaint_grp_auth;
  short           sql_create_not_fenc_auth;
  short           sql_create_not_fenc_grp_auth;
  short           sql_implicit_schema_auth;
  short           sql_implicit_schema_grp_auth;
  short           sql_load_auth;
  short           sql_load_grp_auth;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-AUTHORIZATIONS.
    05 SQL-AUTHORIZATIONS-LEN PIC S9(4) COMP-5.
    05 SQL-SYSADM-AUTH        PIC S9(4) COMP-5.
    05 SQL-DBADM-AUTH         PIC S9(4) COMP-5.
    05 SQL-CREATETAB-AUTH     PIC S9(4) COMP-5.
    05 SQL-BINDADD-AUTH       PIC S9(4) COMP-5.
    05 SQL-CONNECT-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSADM-GRP-AUTH    PIC S9(4) COMP-5.
    05 SQL-DBADM-GRP-AUTH     PIC S9(4) COMP-5.
    05 SQL-CREATETAB-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-BINDADD-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-CONNECT-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-AUTH      PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-GRP-AUTH  PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-AUTH PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-GRP-AUTH PIC S9(4) COMP-5.
*
```

Related reference:

- "sqluadau - Get Authorizations" on page 401

SQL-DIR-ENTRY

This structure is used by the DCS directory APIs.

Table 16. Fields in the SQL-DIR-ENTRY Structure

Field Name	Data Type	Description
STRUCT_ID	SMALLINT	Structure identifier. Set to SQL_DCS_STR_ID (defined in `sqlenv`).
RELEASE	SMALLINT	Release version (assigned by the API).
CODEPAGE	SMALLINT	Code page for comment.
COMMENT	CHAR(30)	Optional description of the database.
LDB	CHAR(8)	Local name of the database; must match database alias in system database directory.
TDB	CHAR(18)	Actual name of the database.
AR	CHAR(32)	Name of the application client.
PARM	CHAR(512)	Contains transaction program prefix, transaction program name, SQLCODE mapping file name, and disconnect and security option.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQL-DIR-ENTRY */
/* ... */
SQL_STRUCTURE sql_dir_entry
{
  unsigned short        struct_id;
  unsigned short        release;
  unsigned short        codepage;
  _SQLOLDCHAR           comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR           ldb[SQL_DBNAME_SZ + 1];
  _SQLOLDCHAR           tdb[SQL_LONG_NAME_SZ + 1];
  _SQLOLDCHAR            ar[SQL_AR_SZ + 1];
  _SQLOLDCHAR            parm[SQL_PARAMETER_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-DIR-ENTRY.
    05 STRUCT-ID                PIC 9(4) COMP-5.
    05 RELEASE-LVL              PIC 9(4) COMP-5.
    05 CODEPAGE                 PIC 9(4) COMP-5.
    05 COMMENT                  PIC X(30).
    05 FILLER                   PIC X.
    05 LDB                      PIC X(8).
    05 FILLER                   PIC X.
    05 TDB                      PIC X(18).
    05 FILLER                   PIC X.
    05 AR                       PIC X(32).
    05 FILLER                   PIC X.
    05 PARM                     PIC X(512).
    05 FILLER                   PIC X.
    05 FILLER                   PIC X(1).
*
```

SQLA-FLAGINFO

This structure is used to hold flagger information.

Table 17. Fields in the SQLA-FLAGINFO Structure

Field Name	Data Type	Description
VERSION	SMALLINT	Input field that must be set to SQLA_FLAG_VERSION (defined in `sqlaprep`).
MSGS	Structure	An imbedded *sqla_flagmsgs* structure.

Table 18. Fields in the SQLA-FLAGMSGS Structure

Field Name	Data Type	Description
COUNT	SMALLINT	Output field set to the number of messages returned by the flagger.
SQLCA	Array	Array of SQLCA structures returning information from the flagger.

Language syntax:

C Structure

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGINFO */
/* ... */
```

```
SQL_STRUCTURE sqla_flaginfo
{
  short            version;
  short            padding;
  struct           sqla_flagmsgs msgs;
};
/* ... */
```

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGMSGS */
/* ... */
SQL_STRUCTURE sqla_flagmsgs
{
  short            count;
  short            padding;
  SQL_STRUCTURE sqlca sqlca[SQLA_FLAG_MAXMSGS];
};
/* ... */
```

COBOL Structure

```
* File: sqlaprep.cbl
01 SQLA-FLAGINFO.
    05 SQLFLAG-VERSION            PIC 9(4) COMP-5.
    05 FILLER                     PIC X(2).
    05 SQLFLAG-MSGS.
        10 SQLFLAG-MSGS-COUNT      PIC 9(4) COMP-5.
        10 FILLER                  PIC X(2).
        10 SQLFLAG-MSGS-SQLCA OCCURS 10 TIMES.
*
```

SQLB-TBS-STATS

This structure is used to return additional table space statistics to an application program.

Table 19. Fields in the SQLB-TBS-STATS Structure

Field Name	Data Type	Description
TOTALPAGES	INTEGER	Total operating system space occupied by the table space (in 4KB pages). For DMS, this is the sum of the container sizes (including overhead). For SMS, this is the sum of all file space used for the tables stored in this table space. This is the only piece of information returned for SMS table spaces; the other fields are set to this value or zero.
USEABLEPAGES	INTEGER	For DMS, equal to TOTALPAGES minus (overhead plus partial extents). For SMS, equal to TOTALPAGES.
USEDPAGES	INTEGER	For DMS, the total number of pages in use. For SMS, equal to TOTALPAGES.

Table 19. Fields in the SQLB-TBS-STATS Structure (continued)

Field Name	Data Type	Description
FREEPAGES	INTEGER	For DMS, equal to USEABLEPAGES minus USEDPAGES. For SMS, not applicable.
HIGHWATERMARK	INTEGER	For DMS, the high water mark is the current ″end″ of the table space address space. In other words, the page number of the first free extent following the last allocated extent of a table space. Note that this is not really a ″high water mark″, but rather a ″current water mark″, since the value can decrease. For SMS, this is not applicable.

During a table space rebalance, the number of useable pages will include pages for the newly added container, but these new pages will not be reflected in the number of free pages until the rebalance is complete. When a table space rebalance is *not* taking place, the number of used pages plus the number of free pages will equal the number of useable pages.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLB-TBS-STATS */
/* ... */
SQL_STRUCTURE SQLB_TBS_STATS
{
  sqluint32   totalPages;
  sqluint32   useablePages;
  sqluint32   usedPages;
  sqluint32   freePages;
  sqluint32   highWaterMark;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLB-TBS-STATS.
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-USED-PAGES         PIC 9(9) COMP-5.
    05 SQL-FREE-PAGES         PIC 9(9) COMP-5.
    05 SQL-HIGH-WATER-MARK    PIC 9(9) COMP-5.
*
```

SQLB-TBSCONTQRY-DATA

This structure is used to return container data to an application program.

Table 20. Fields in the SQLB-TBSCONTQRY-DATA Structure

Field Name	Data Type	Description
ID	INTEGER	Container identifier.
NTBS	INTEGER	Always 1.
TBSID	INTEGER	Table space identifier.
NAMELEN	INTEGER	Length of the container name (for languages other than C).
NAME	CHAR(256)	Container name.
UNDERDBDIR	INTEGER	Either 1 (container is under the DB directory) or 0 (container is not under the DB directory).
CONTTYPE	INTEGER	Container type.
TOTALPAGES	INTEGER	Total number of pages occupied by the table space container.
USEABLEPAGES	INTEGER	For DMS, TOTALPAGES minus overhead. For SMS, equal to TOTALPAGES.

Table 20. Fields in the SQLB-TBSCONTQRY-DATA Structure (continued)

Field Name	Data Type	Description
OK	INTEGER	Either 1 (container is accessible) or 0 (container is inaccessible). Zero indicates an abnormal situation that usually requires the attention of the database administrator.

Possible values for *CONTTYPE* (defined in `sqlutil`) are:

SQLB_CONT_PATH
Specifies a directory path (SMS only).

SQLB_CONT_DISK
Specifies a raw device (DMS only).

SQLB_CONT_FILE
Specifies a file (DMS only).

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLB-TBSCONTQRY-DATA */
/* ... */
SQL_STRUCTURE SQLB_TBSCONTQRY_DATA
{
  sqluint32   id;
  sqluint32   nTbs;
  sqluint32   tbsID;
  sqluint32   nameLen;
  char        name[SQLB_MAX_CONTAIN_NAME_SZ];
  sqluint32   underDBDir;
  sqluint32   contType;
  sqluint32   totalPages;
  sqluint32   useablePages;
  sqluint32   ok;
};
/* ... */
```

COBOL Structure

```
* File: sqlutbcq.cbl
01 SQLB-TBSCONTQRY-DATA.
    05 SQL-ID                PIC 9(9) COMP-5.
    05 SQL-N-TBS             PIC 9(9) COMP-5.
    05 SQL-TBS-ID            PIC 9(9) COMP-5.
    05 SQL-NAME-LEN          PIC 9(9) COMP-5.
    05 SQL-NAME              PIC X(256).
    05 SQL-UNDER-DBDIR       PIC 9(9) COMP-5.
    05 SQL-CONT-TYPE         PIC 9(9) COMP-5.
    05 SQL-TOTAL-PAGES       PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES     PIC 9(9) COMP-5.
    05 SQL-OK                PIC 9(9) COMP-5.
*
```

SQLB-TBSPQRY-DATA

This structure is used to return table space data to an application program.

Table 21. Fields in the SQLB-TBSPQRY-DATA Structure

Field Name	Data Type	Description
TBSPQVER	CHAR(8)	Structure version identifier.
ID	INTEGER	Internal identifier for the table space.
NAMELEN	INTEGER	Length of the table space name.
NAME	CHAR(128)	Null-terminated name of the table space.
TOTALPAGES	INTEGER	Number of pages specified by CREATE TABLESPACE (DMS only).

Table 21. Fields in the SQLB-TBSPQRY-DATA Structure (continued)

Field Name	Data Type	Description
USEABLEPAGES	INTEGER	TOTALPAGES minus overhead (DMS only). This value is rounded down to the next multiple of 4KB.
FLAGS	INTEGER	Bit attributes for the table space.
PAGESIZE	INTEGER	Page size (in bytes) of the table space. Currently fixed at 4KB.
EXTSIZE	INTEGER	Extent size (in pages) of the table space.
PREFETCHSIZE	INTEGER	Prefetch size.
NCONTAINERS	INTEGER	Number of containers in the table space.
TBSSTATE	INTEGER	Table space states.
LIFELSN	CHAR(6)	Time stamp identifying the origin of the table space.
FLAGS2	INTEGER	Bit attributes for the table space.
MINIMUMRECTIME	CHAR(27)	Earliest point in time that may be specified by point-in-time table space rollforward.
STATECHNGOBJ	INTEGER	If TBSSTATE is SQLB_LOAD_PENDING or SQLB_DELETE_PENDING, the object ID in table space STATECHANGEID that caused the table space state to be set. Otherwise zero.
STATECHNGID	INTEGER	If TBSSTATE is SQLB_LOAD_PENDING or SQLB_DELETE_PENDING, the table space ID of the object STATECHANGEOBJ that caused the table space state to be set. Otherwise zero.
NQUIESCERS	INTEGER	If TBSSTATE is SQLB_QUIESCED_SHARE, UPDATE, or EXCLUSIVE, the number of quiescers of the table space and the number of entries in QUIESCERS.
QUIESCEID	INTEGER	The table space ID of the object QUIESCEOBJ that caused the table space to be quiesced.
QUIESCEOBJ	INTEGER	The object ID in table space QUIESCEID that caused the table space to be quiesced.
RESERVED	CHAR(32)	Reserved for future use.

Possible values for *FLAGS* (defined in `sqlutil`) are:

SQLB_TBS_SMS
System Managed Space

SQLB_TBS_DMS
Database Managed Space

SQLB_TBS_ANY
Regular contents

SQLB_TBS_LONG
Long field data

SQLB_TBS_SYSTMP
System temporary data.

SQLB_TBS_USRTMP
User temporary data.

Possible values for *TBSSTATE* (defined in `sqlutil`) are:

SQLB_NORMAL
Normal

SQLB_QUIESCED_SHARE
Quiesced: SHARE

SQLB_QUIESCED_UPDATE
Quiesced: UPDATE

SQLB_QUIESCED_EXCLUSIVE
Quiesced: EXCLUSIVE

SQLB_LOAD_PENDING
Load pending

SQLB_DELETE_PENDING
Delete pending

SQLB_BACKUP_PENDING
Backup pending

SQLB_ROLLFORWARD_IN_PROGRESS
Roll forward in progress

SQLB_ROLLFORWARD_PENDING
Roll forward pending

SQLB_RESTORE_PENDING
Restore pending

SQLB_DISABLE_PENDING
Disable pending

SQLB_REORG_IN_PROGRESS
Reorganization in progress

SQLB_BACKUP_IN_PROGRESS
Backup in progress

SQLB_STORDEF_PENDING
Storage must be defined

SQLB_RESTORE_IN_PROGRESS
Restore in progress

SQLB_STORDEF_ALLOWED
Storage may be defined

SQLB_STORDEF_FINAL_VERSION
Storage definition is in ′final′ state

SQLB_STORDEF_CHANGED
Storage definition was changed prior to roll forward

SQLB_REBAL_IN_PROGRESS
DMS rebalancer is active

SQLB_PSTAT_DELETION
Table space deletion in progress

SQLB_PSTAT_CREATION
Table space creation in progress.

Possible values for *FLAGS2* (defined in `sqlutil`) are:

SQLB_STATE_SET
For service use only.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_TBSPQRY_DATA
{
   char                        tbspqver[SQLB_SVERSION_SIZE];
   sqluint32                   id;
   sqluint32                   nameLen;
   char                        name[SQLB_MAX_TBS_NAME_SZ];
   sqluint32                   totalPages;
   sqluint32                   useablePages;
   sqluint32                   flags;
   sqluint32                   pageSize;
   sqluint32                   extSize;
   sqluint32                   prefetchSize;
   sqluint32                   nContainers;
   sqluint32                   tbsState;
   char                        lifeLSN[6];
   char                        pad[2];
   sqluint32                   flags2;
   char                        minimumRecTime[SQL_STAMP_STRLEN+1];
   char                        pad1[1];
   sqluint32                   StateChngObj;
   sqluint32                   StateChngID;
   sqluint32                   nQuiescers;
   struct SQLB_QUIESCER_DATA   quiescer[SQLB_MAX_QUIESCERS];
   char                        reserved[32];
};
/* ... */
```

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_QUIESCER_DATA
{
   sqluint32   quiesceId;
   sqluint32   quiesceObject;
};
/* ... */
```

COBOL Structure

```
* File: sqlutbsp.cbl
01 SQLB-TBSPQRY-DATA.
    05 SQL-TBSPQVER           PIC X(8).
    05 SQL-ID                 PIC 9(9) COMP-5.
    05 SQL-NAME-LEN           PIC 9(9) COMP-5.
    05 SQL-NAME               PIC X(128).
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-FLAGS              PIC 9(9) COMP-5.
    05 SQL-PAGE-SIZE          PIC 9(9) COMP-5.
    05 SQL-EXT-SIZE           PIC 9(9) COMP-5.
    05 SQL-PREFETCH-SIZE      PIC 9(9) COMP-5.
    05 SQL-N-CONTAINERS       PIC 9(9) COMP-5.
    05 SQL-TBS-STATE          PIC 9(9) COMP-5.
    05 SQL-LIFE-LSN           PIC X(6).
    05 SQL-PAD                PIC X(2).
    05 SQL-FLAGS2             PIC 9(9) COMP-5.
    05 SQL-MINIMUM-REC-TIME   PIC X(26).
    05 FILLER                 PIC X.
    05 SQL-PAD1               PIC X(1).
    05 SQL-STATE-CHNG-OBJ     PIC 9(9) COMP-5.
    05 SQL-STATE-CHNG-ID      PIC 9(9) COMP-5.
    05 SQL-N-QUIESCERS        PIC 9(9) COMP-5.
    05 SQL-QUIESCER OCCURS 5 TIMES.
        10 SQL-QUIESCE-ID     PIC 9(9) COMP-5.
        10 SQL-QUIESCE-OBJECT PIC 9(9) COMP-5.
    05 SQL-RESERVED           PIC X(32).
*
```

SQLCA

The SQL communications area (SQLCA) structure is used by the database manager to return error information to an application program. This structure is updated after every API call and SQL statement issued.

Language syntax:

C Structure

```
/* File: sqlca.h */
/* Structure: SQLCA */
/* ... */
SQL_STRUCTURE  sqlca
{
  _SQLOLDCHAR    sqlcaid[8];
  sqlint32       sqlcabc;
  #ifdef DB2_SQL92E
  sqlint32       sqlcade;
  #else
  sqlint32       sqlcode;
  #endif
  short          sqlerrml;
  _SQLOLDCHAR    sqlerrmc[70];
  _SQLOLDCHAR    sqlerrp[8];
  sqlint32       sqlerrd[6];
  _SQLOLDCHAR    sqlwarn[11];
  #ifdef DB2_SQL92E
  _SQLOLDCHAR    sqlstat[5];
  #else
  _SQLOLDCHAR    sqlstate[5];
  #endif
};
/* ... */
```

COBOL Structure

```
* File: sqlca.cbl
01 SQLCA SYNC.
    05 SQLCAID PIC X(8) VALUE "SQLCA   ".
    05 SQLCABC PIC S9(9) COMP-5 VALUE 136.
    05 SQLCODE PIC S9(9) COMP-5.
    05 SQLERRM.
    05 SQLERRP PIC X(8).
    05 SQLERRD OCCURS 6 TIMES PIC S9(9) COMP-5.
    05 SQLWARN.
        10 SQLWARN0 PIC X.
        10 SQLWARN1 PIC X.
        10 SQLWARN2 PIC X.
        10 SQLWARN3 PIC X.
        10 SQLWARN4 PIC X.
        10 SQLWARN5 PIC X.
        10 SQLWARN6 PIC X.
        10 SQLWARN7 PIC X.
        10 SQLWARN8 PIC X.
        10 SQLWARN9 PIC X.
        10 SQLWARNA PIC X.
    05 SQLSTATE PIC X(5).
*
```

Related reference:

- "SQLCA (SQL communications area)" in the *SQL Reference, Volume 1*

SQLCHAR

This structure is used to pass variable length data to the database manager.

Table 22. Fields in the SQLCHAR Structure

Field Name	Data Type	Description
LENGTH	SMALLINT	Length of the character string pointed to by *DATA*.
DATA	CHAR(n)	An array of characters of length *LENGTH*.

Language syntax:

C Structure

```
/* File: sql.h */
/* Structure: SQLCHAR */
/* ... */
SQL_STRUCTURE sqlchar
{
  short                 length;
  _SQLOLDCHAR           data[1];
};
/* ... */
```

SQLCHAR

COBOL Structure

This is not defined in any header file. The following is an example showing how it can be done:

```
* Replace maxlen with the appropriate value:
01 SQLCHAR.
49 SQLCHAR-LEN  PIC S9(4) COMP-5.
49 SQLCHAR-DATA PIC X(maxlen).
```

SQLDA

The SQL descriptor area (SQLDA) structure is a collection of variables that is required for execution of the SQL DESCRIBE statement. The SQLDA variables are options that can be used with the PREPARE, OPEN, FETCH, EXECUTE, and CALL statements.

An SQLDA communicates with dynamic SQL; it can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH statement.

SQLDAs are supported for all languages, but predefined declarations are provided only for C, REXX, FORTRAN, and COBOL.

The meaning of the information in an SQLDA depends on its use. In PREPARE and DESCRIBE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, FETCH, and CALL, an SQLDA describes host variables.

Language syntax:

C Structure

```
/* File: sqlda.h */
/* Structure: SQLDA */
/* ... */
SQL_STRUCTURE  sqlda
{
  _SQLOLDCHAR    sqldaid[8];
  long           sqldabc;
  short          sqln;
  short          sqld;
  struct sqlvar  sqlvar[1];
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLVAR */
/* ... */
SQL_STRUCTURE  sqlvar
{
  short          sqltype;
  short          sqllen;
  _SQLOLDCHAR   *SQL_POINTER sqldata;
  short         *SQL_POINTER sqlind;
  struct sqlname sqlname;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLNAME */
/* ... */
SQL_STRUCTURE  sqlname
{
  short          length;
  _SQLOLDCHAR    data[30];
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLVAR2 */
/* ... */
SQL_STRUCTURE  sqlvar2
{
  union sql8bytelen len;
  char *SQL_POINTER sqldatalen;
  struct sqldistinct_type sqldatatype_name;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQL8BYTELEN */
/* ... */
union sql8bytelen
{
  long        reserve1[2];
  long        sqllonglen;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLDISTINCT-TYPE */
/* ... */
SQL_STRUCTURE  sqldistinct_type
{
  short          length;
  char           data[27];
  char           reserved1[3];
};
/* ... */
```

COBOL Structure

```
* File: sqlda.cbl
01 SQLDA SYNC.
    05 SQLDAID PIC X(8) VALUE "SQLDA  ".
    05 SQLDABC PIC S9(9) COMP-5.
    05 SQLN PIC S9(4) COMP-5.
    05 SQLD PIC S9(4) COMP-5.
    05 SQLVAR-ENTRIES OCCURS 0 TO 1489 TIMES
        10 SQLVAR.
        10 SQLVAR2 REDEFINES SQLVAR.
*
```

Related reference:

- "SQLDA (SQL descriptor area)" in the *SQL Reference, Volume 1*

SQLDCOL

This structure is used to pass variable column information to the sqluexpr, sqluimpr, and db2Load APIs.

Table 23. Fields in the SQLDCOL Structure

Field Name	**Data Type**	**Description**
DCOLMETH	SMALLINT	A character indicating the method to be used to select columns within the data file.
DCOLNUM	SMALLINT	The number of columns specified in the array *DCOLNAME*.
DCOLNAME	Array	An array of *DCOLNUM sqldcoln* structures.

Table 24. Fields in the SQLDCOLN Structure

Field Name	**Data Type**	**Description**
DCOLNLEN	SMALLINT	Length of the data pointed to by *DCOLNPTR*.
DCOLNPTR	Pointer	Pointer to a data element determined by *DCOLMETH*.
Note: The *DCOLNLEN* and *DCOLNPTR* fields are repeated for each column specified.		

Table 25. Fields in the SQLLOCTAB Structure

Field Name	**Data Type**	**Description**
LOCPAIR	Array	An array of *sqllocpair* structures.

Table 26. Fields in the SQLLOCPAIR Structure

Field Name	**Data Type**	**Description**
BEGIN_LOC	SMALLINT	Starting position of the column data in the external file.
END_LOC	SMALLINT	Ending position of the column data in the external file.

The valid values for *DCOLMETH* (defined in `sqlutil`) are:

SQL_METH_N
Names. When importing or loading, use the column names provided via this structure to identify the data to import or load from the external file. The case of these column names must match the case of the corresponding names in the system catalogs. When exporting, use the column names provided via this structure as the column names in the output file.

The *dcolnptr* pointer of each element of the *dcolname* array points to an array of characters, of length *dcolnlen* bytes, that make up the name of a column to be imported or loaded. The *dcolnum* field, which must be positive, indicates the number of elements in the *dcolname* array.

This method is invalid if the external file does not contain column names (DEL or ASC format files, for example).

SQL_METH_P
Positions. When importing or loading, use starting column positions provided via this structure to identify the data to import or load from the external file. This method is not valid when exporting data.

The *dcolnptr* pointer of each element of the *dcolname* array is ignored, while the *dcolnlen* field contains a column position in the external file. The *dcolnum* field, which must be positive, indicates the number of elements in the *dcolname* array.

The lowest valid column position value is 1 (indicating the first column), and the highest valid value depends on the external file type. Positional selection is not valid for import of ASC files.

SQL_METH_L
Locations. When importing or loading, use starting and ending column positions provided via this structure to identify the data to import or load from the external file. This method is not valid when exporting data.

The *dcolnptr* field of the first element of the *dcolname* array points to an *sqlloctab* structure, which consists of an array of *sqllocpair* structures. The number of elements in this array is determined by the *dcolnum* field of the *sqldcol* structure, which must be positive. Each element in the array is a pair of 2-byte integers that indicate where the column begins and ends. The first element of each location pair is the byte within the file where the column begins, and the second element is the byte where the column ends. The first byte position within a row in the file is considered byte position 1. The columns can overlap.

SQL_METH_D

Default. When importing or loading DEL and IXF files, the first column of the file is loaded or imported into the first column of the table, and so on. When importing or loading ASC files, the selection of columns is in a file where the name of which is included in the file type modifier POSITIONSFILE. When exporting, the default names are used for the columns in the external file.

The *dcolnum* and *dcolname* fields of the *sqldcol* structure are both ignored, and the columns from the external file are taken in their natural order.

A column from the external file can be used in the array more than once. It is not necessary to use every column from the external file.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLDCOL */
/* ... */
SQL_STRUCTURE sqldcol
{
  short           dcolmeth;
  short           dcolnum;
  struct sqldcoln dcolname[1];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLDCOLN */
/* ... */
SQL_STRUCTURE sqldcoln
{
  short           dcolnlen;
  char            *dcolnptr;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLLOCTAB */
/* ... */
SQL_STRUCTURE sqlloctab
{
  struct sqllocpair locpair[1];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLLOCPAIR */
/* ... */
SQL_STRUCTURE sqllocpair
{
  short           begin_loc;
  short           end_loc;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-DCOLDATA.
    05 SQL-DCOLMETH           PIC S9(4) COMP-5.
    05 SQL-DCOLNUM            PIC S9(4) COMP-5.
    05 SQLDCOLN OCCURS 0 TO 255 TIMES DEPENDING ON SQL-DCOLNUM.
        10 SQL-DCOLNLEN       PIC S9(4) COMP-5.
        10 FILLER             PIC X(2).
        10 SQL-DCOLN-PTR      USAGE IS POINTER.
*
* File: sqlutil.cbl
01 SQL-LOCTAB.
    05 SQL-LOC-PAIR OCCURS 1 TIMES.
        10 SQL-BEGIN-LOC      PIC S9(4) COMP-5.
        10 SQL-END-LOC        PIC S9(4) COMP-5.
*
```

Related reference:

- "sqluexpr - Export" on page 408
- "sqluimpr - Import" on page 424

SQLE-ADDN-OPTIONS

This structure is used to pass information to the sqleaddn API.

Table 27. Fields in the SQLE-NODE-APPN Structure

Field Name	Data Type	Description
SQLADDID	CHAR	An "eyecatcher" value which must be set to `SQLE_ADDOPTID_V51`.
TBLSPACE_TYPE	sqluint32	Specifies the type of system temporary table space definitions to be used for the node being added. See below for values.
TBLSPACE_NODE	SQL_PDB_NODE_TYPE	Specifies the node number from which the system temporary table space definitions should be obtained. The node number must exist in the `db2nodes.cfg` file, and is only used if the *tblspace_type* field is set to `SQLE_TABLESPACES_LIKE_NODE`.

Valid values for *TBLSPACE_TYPE* (defined in `sqlenv`) are:

SQLE_TABLESPACES_NONE
Do not create any system temporary table spaces.

SQLE_TABLESPACES_LIKE_NODE
The containers for the system temporary table spaces should be the same as those for the specified node.

SQLE_TABLESPACES_LIKE_CATALOG
The containers for the system temporary table spaces should be the same as those for the catalog node of each database.

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-ADDN-OPTIONS */
/* ... */
SQL_STRUCTURE sqle_addn_options
{
  char                  sqladdid[8];
  sqluint32             tblspace_type;
  SQL_PDB_NODE_TYPE     tblspace_node;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-ADDN-OPTIONS.
    05 SQLADDID                 PIC X(8).
    05 SQL-TBLSPACE-TYPE        PIC 9(9) COMP-5.
    05 SQL-TBLSPACE-NODE        PIC S9(4) COMP-5.
    05 FILLER                   PIC X(2).
*
```

Related reference:
- "sqleaddn - Add Node" on page 296

SQLE-CLIENT-INFO

This structure is used to pass information to the sqleseti and sqleqryi APIs.

This structure specifies:
- The type of information being set or queried
- The length of the data being set or queried
- A pointer to either:
  - An area that will contain the data being set
  - An area of sufficient length to contain the data being queried

Applications can specify the following types of information:

- Client user ID being set or queried. A maximum of 255 characters can be set, although servers can truncate this to some platform-specific value.

  Note: This user ID is for identification purposes only, and is not used for any authorization.
- Client workstation name being set or queried. A maximum of 255 characters can be set, although servers can truncate this to some platform-specific value.
- Client application name being set or queried. A maximum of 255 characters can be set, although servers can truncate this to some platform-specific value.
- Client accounting string being set or queried. A maximum of 200 characters can be set, although servers can truncate this to some platform-specific value.

  Note: The information can be set using the sqlesact API. However, sqlesact does not permit the accounting string to be changed once a connection exists, whereas sqleseti allows the accounting information to be changed for future, as well as already established, connections.

Table 28. Fields in the SQLE-CLIENT-INFO Structure

Field Name	Data Type	Description
TYPE	sqlint32	Setting type.
LENGTH	sqlint32	Length of the value. On sqleseti calls, the length can be between zero and the maximum length defined for the type. A length of zero indicates a null value. On sqleqryi calls, the length is returned, but the area pointed to by *pValue* must be large enough to contain the maximum length for the type. A length of zero indicates a null value.
PVALUE	Pointer	Pointer to an application-allocated buffer that contains the specified value. The data type of this value is dependent on the type field.

The valid entries for the SQLE-CLIENT-INFO TYPE element and the associated descriptions for each entry are listed below:

Table 29. Connection Settings

Type	Data Type	Description
SQLE_CLIENT_INFO_USERID	CHAR(255)	The user ID for the client. Some servers may truncate the value. For example, DB2 for OS/390 servers support up to length 16. This user ID is for identification purposes only, and is not used for any authorization.
SQLE_CLIENT_INFO_ WRKSTNNAME	CHAR(255)	The workstation name for the client. Some servers may truncate the value. For example, DB2 for OS/390 servers support up to length 18.
SQLE_CLIENT_INFO_ APPLNAME	CHAR(255)	The application name for the client. Some servers may truncate the value. For example, DB2 for OS/390 servers support up to length 32.
SQLE_CLIENT_INFO_ ACCTSTR	CHAR(200)	The accounting string for the client. Some servers may truncate the value. For example, DB2 for OS/390 servers support up to length 200.
Note: These field names are defined for the C programming language. There are similar names for FORTRAN and COBOL, which have the same semantics.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-CLIENT-INFO */
/* ... */
SQL_STRUCTURE sqle_client_info
{
  unsigned short         type;
  unsigned short         length;
  char                   *pValue;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-CLIENT-INFO.
    05 SQLE-CLIENT-INFO-ITEM OCCURS 4 TIMES.
```

```
        10 SQLE-CLIENT-INFO-TYPE   PIC S9(4) COMP-5.
        10 SQLE-CLIENT-INFO-LENGTH PIC S9(4) COMP-5.
        10 SQLE-CLIENT-INFO-VALUE  USAGE IS POINTER.
*
```

Related reference:

- "sqlesact - Set Accounting String" on page 382
- "sqleseti - Set Client Information" on page 389
- "sqleqryi - Query Client Information" on page 378

SQLE-CONN-SETTING

This structure is used to specify connection setting types and values for the sqleqryc and sqlesetc APIs.

Table 30. Fields in the SQLE-CONN-SETTING Structure

Field Name	Data Type	Description
TYPE	SMALLINT	Setting type.
VALUE	SMALLINT	Setting value.

The valid entries for the SQLE-CONN-SETTING TYPE element and the associated descriptions for each entry are listed below (defined in `sqlenv` and `sql`):

Table 31. Connection Settings

Type	Value	Description
SQL_CONNECT_TYPE	SQL_CONNECT_1	Type 1 CONNECTs enforce the single database per unit of work semantics of older releases, also known as the rules for remote unit of work (RUOW).
	SQL_CONNECT_2	Type 2 CONNECTs support the multiple databases per unit of work semantics of DUOW.
SQL_RULES	SQL_RULES_DB2	Enable the SQL CONNECT statement to switch the current connection to an established (dormant) connection.
	SQL_RULES_STD	Permit only the establishment of a new connection through the SQL CONNECT statement. The SQL SET CONNECTION statement must be used to switch the current connection to a dormant connection.

Table 31. Connection Settings (continued)

Type	Value	Description
SQL_DISCONNECT	SQL_DISCONNECT_EXPL	Removes those connections that have been explicitly marked for release by the SQL RELEASE statement at commit.
	SQL_DISCONNECT_COND	Breaks those connections that have no open WITH HOLD cursors at commit, and those that have been marked for release by the SQL RELEASE statement.
	SQL_DISCONNECT_AUTO	Breaks all connections at commit.
SQL_SYNCPOINT	SQL_SYNC_TWOPHASE	Requires a Transaction Manager (TM) to coordinate two-phase commits among databases that support this protocol.
	SQL_SYNC_ONEPHASE	Uses one-phase commits to commit the work done by each database in multiple database transactions. Enforces single updater, multiple read behavior.
	SQL_SYNC_NONE	Uses one-phase commits to commit work done, but does not enforce single updater, multiple read behavior.
SQL_MAX_NETBIOS_ CONNECTIONS	Between 1 and 254	This specifies the maximum number of concurrent connections that can be made using a NETBIOS adapter in an application.
SQL_DEFERRED_PREPARE	SQL_DEFERRED_PREPARE_ NO	The PREPARE statement will be executed at the time it is issued.

Table 31. Connection Settings (continued)

Type	Value	Description
	SQL_DEFERRED_PREPARE_ YES	Execution of the PREPARE statement will be deferred until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued. The PREPARE statement will not be deferred if it uses the INTO clause, which requires an SQLDA to be returned immediately. However, if the PREPARE INTO statement is issued for a cursor that does not use any parameter markers, the processing will be optimized by pre-OPENing the cursor when the PREPARE is executed.
	SQL_DEFERRED_PREPARE_ ALL	Same as YES, except that a PREPARE INTO statement which contains parameter markers *is* deferred. If a PREPARE INTO statement does not contain parameter markers, pre-OPENing of the cursor will still be performed. If the PREPARE statement uses the INTO clause to return an SQLDA, the application must not reference the content of this SQLDA until the OPEN, DESCRIBE, or EXECUTE statement is issued and returned.
SQL_CONNECT_NODE	Between 0 and 999, or the keyword SQL_CONN_CATALOG_ NODE.	Specifies the node to which a connect is to be made. Overrides the value of the environment variable **DB2NODE**. For example, if nodes 1, 2, and 3 are defined, the client only needs to be able to access one of these nodes. If only node 1 containing databases has been cataloged, and this parameter is set to 3, the next connect attempt will result in a connection at node 3, after an initial connection at node 1.

Table 31. Connection Settings (continued)

Type	Value	Description
SQL_ATTACH_NODE	Between 0 and 999.	Specifies the node to which an attach is to be made. Overrides the value of the environment variable **DB2NODE**. For example, if nodes 1, 2, and 3 are defined, the client only needs to be able to access one of these nodes. If only node 1 containing databases has been cataloged, and this parameter is set to 3, then the next attach attempt will result in an attachment at node 3, after an initial attachment at node 1.
Note: These field names are defined for the C programming language. There are similar names for FORTRAN and COBOL, which have the same semantics.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-CONN-SETTING */
/* ... */
SQL_STRUCTURE sqle_conn_setting
{
  unsigned short         type;
  unsigned short         value;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-CONN-SETTING.
    05 SQLE-CONN-SETTING-ITEM OCCURS 7 TIMES.
        10 SQLE-CONN-TYPE  PIC S9(4) COMP-5.
        10 SQLE-CONN-VALUE PIC S9(4) COMP-5.
*
```

Related reference:

- "sqlesetc - Set Client" on page 386
- "sqleqryc - Query Client" on page 376

SQLE-NODE-APPC

This structure is used to catalog APPC nodes for the sqlectnd API.

Table 32. Fields in the SQLE-NODE-APPC Structure

Field Name	Data Type	Description
LOCAL_LU	CHAR(8)	Local_lu name.
PARTNER_LU	CHAR(8)	Alias Partner_lu name.
MODE	CHAR(8)	Mode.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPC */
/* ... */
SQL_STRUCTURE sqle_node_appc
{
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-APPC.
    05 LOCAL-LU             PIC X(8).
    05 FILLER               PIC X.
    05 PARTNER-LU           PIC X(8).
    05 FILLER               PIC X.
    05 TRANS-MODE           PIC X(8).
    05 FILLER               PIC X.
*
```

Related reference:

- “sqlectnd - Catalog Node” on page 322

SQLE-NODE-APPN

This structure is used to catalog APPN nodes for the sqlectnd API.

Table 33. Fields in the SQLE-NODE-APPN Structure

Field Name	Data Type	Description
NETWORKID	CHAR(8)	Network ID.
REMOTE_LU	CHAR(8)	Alias Remote_lu name.
LOCAL_LU	CHAR(8)	Alias Local_lu name.
MODE	CHAR(8)	Mode.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPN */
/* ... */
SQL_STRUCTURE sqle_node_appn
{
  _SQLOLDCHAR    networkid[SQL_NETID_SZ + 1];
  _SQLOLDCHAR    remote_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-APPN.
    05 NETWORKID                 PIC X(8).
    05 FILLER                    PIC X.
    05 REMOTE-LU                 PIC X(8).
    05 FILLER                    PIC X.
    05 LOCAL-LU                  PIC X(8).
    05 FILLER                    PIC X.
    05 TRANS-MODE                PIC X(8).
    05 FILLER                    PIC X.
*
```

Related reference:

- “sqlectnd - Catalog Node” on page 322

SQLE-NODE-CPIC

This structure is used to catalog CPIC nodes for the sqlectnd API.

Table 34. Fields in the SQLE-NODE-CPIC Structure

Field Name	Data Type	Description
SYM_DEST_NAME	CHAR(8)	Symbolic destination name of remote partner.
SECURITY_TYPE	SMALLINT	Security type.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Valid values for *SECURITY_TYPE* (defined in `sqlenv`) are:

SQL_CPIC_SECURITY_NONE

SQL_CPIC_SECURITY_SAME

SQL_CPIC_SECURITY_PROGRAM

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-CPIC */
/* ... */
SQL_STRUCTURE sqle_node_cpic
{
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ+1];
  unsigned short security_type;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-CPIC.
    05 SYM-DEST-NAME          PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
    05 SECURITY-TYPE          PIC 9(4) COMP-5.
*
```

Related reference:

- “sqlectnd - Catalog Node” on page 322

SQLE-NODE-IPXSPX

This structure is used to catalog IPX/SPX nodes for the sqlectnd API.

Table 35. Fields in the SQLE-NODE-IPXSPX Structure

Field Name	Data Type	Description
FILESERVER	CHAR(48)	Name of the NetWare file server where the DB2 server instance is registered.
OBJECTNAME	CHAR(48)	The database manager server instance is represented as the object, *objectname*, on the NetWare file server. The server's IPX/SPX internetwork address is stored and retrieved from this object.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-IPXSPX */
/* ... */
SQL_STRUCTURE sqle_node_ipxspx
{
  char           fileserver[SQL_FILESERVER_SZ+1];
  char           objectname[SQL_OBJECTNAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-IPXSPX.
    05 SQL-FILESERVER        PIC X(48).
    05 FILLER                PIC X.
    05 SQL-OBJECTNAME        PIC X(48).
    05 FILLER                PIC X.
*
```

Related reference:

- "sqlectnd - Catalog Node" on page 322

SQLE-NODE-LOCAL

This structure is used to catalog local nodes for the sqlectnd API.

Table 36. Fields in the SQLE-NODE-LOCAL Structure

Field Name	Data Type	Description
INSTANCE_NAME	CHAR(8)	Name of an instance.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-LOCAL */
/* ... */
SQL_STRUCTURE sqle_node_local
{
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-LOCAL.
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 FILLER                 PIC X.
*
```

Related reference:

- "sqlectnd - Catalog Node" on page 322

SQLE-NODE-NETB

This structure is used to catalog NetBIOS nodes for the sqlectnd API.

Table 37. Fields in the SQLE-NODE-NETB Structure

Field Name	Data Type	Description
ADAPTER	SMALLINT	Local LAN adapter.
REMOTE_NNAME	CHAR(8)	*Nname* of the remote workstation that is stored in the database manager configuration file on the server instance.

Table 37. Fields in the SQLE-NODE-NETB Structure (continued)

Field Name	Data Type	Description
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NETB */
/* ... */
SQL_STRUCTURE sqle_node_netb
{
  unsigned short adapter;
  _SQLOLDCHAR    remote_nname[SQL_RMTLU_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-NETB.
    05 ADAPTER                PIC 9(4) COMP-5.
    05 REMOTE-NNAME           PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
*
```

Related reference:

- "sqlectnd - Catalog Node" on page 322

SQLE-NODE-NPIPE

This structure is used to catalog named pipe nodes for the sqlectnd API.

Table 38. Fields in the SQLE-NODE-NPIPE Structure

Field Name	Data Type	Description
COMPUTERNAME	CHAR(15)	Computer name.
INSTANCE_NAME	CHAR(8)	Name of an instance.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NPIPE */
/* ... */
SQL_STRUCTURE sqle_node_npipe
{
  char           computername[SQL_COMPUTERNAME_SZ+1];
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-NPIPE.
    05 COMPUTERNAME          PIC X(15).
    05 FILLER                PIC X.
    05 INSTANCE-NAME         PIC X(8).
    05 FILLER                PIC X.
*
```

Related reference:

- "sqlectnd - Catalog Node" on page 322

SQLE-NODE-STRUCT

This structure is used to catalog nodes for the sqlectnd API.

Table 39. Fields in the SQLE-NODE-STRUCT Structure

Field Name	**Data Type**	**Description**
STRUCT_ID	SMALLINT	Structure identifier.
CODEPAGE	SMALLINT	Code page for comment.
COMMENT	CHAR(30)	Optional description of the node.
NODENAME	CHAR(8)	Local name for the node where the database is located.
PROTOCOL	CHAR(1)	Communications protocol type.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Valid values for *PROTOCOL* (defined in `sqlenv`) are:

SQL_PROTOCOL_APPC

SQL_PROTOCOL_APPN

SQL_PROTOCOL_CPIC

SQL_PROTOCOL_IPXSPX

SQL_PROTOCOL_LOCAL

SQL_PROTOCOL_NETB

SQL_PROTOCOL_NPIPE

SQL_PROTOCOL_SOCKS

SQL_PROTOCOL_TCPIP

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-STRUCT */
/* ... */
SQL_STRUCTURE sqle_node_struct
{
  unsigned short struct_id;
  unsigned short codepage;
  _SQLOLDCHAR    comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ + 1];
  unsigned char  protocol;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-STRUCT.
    05 STRUCT-ID                 PIC 9(4) COMP-5.
    05 CODEPAGE                  PIC 9(4) COMP-5.
    05 COMMENT                   PIC X(30).
    05 FILLER                    PIC X.
    05 NODENAME                  PIC X(8).
    05 FILLER                    PIC X.
    05 PROTOCOL                  PIC X.
    05 FILLER                    PIC X(1).
*
```

Related reference:

- “sqlectnd - Catalog Node” on page 322

SQLE-NODE-TCPIP

This structure is used to catalog TCP/IP nodes for the sqlectnd API.

Note: To catalog a TCP/IP SOCKS node, set the PROTOCOL type in the node directory structure to `SQL_PROTOCOL_SOCKS` in the *SQLE-NODE-STRUCT* structure before calling the sqlectnd API.

Table 40. Fields in the SQLE-NODE-TCPIP Structure

Field Name	Data Type	Description
HOSTNAME	CHAR(255)	The name of the TCP/IP host on which the DB2 server instance resides.
SERVICE_NAME	CHAR(14)	TCP/IP service name or associated port number of the DB2 server instance.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-TCPIP */
/* ... */
SQL_STRUCTURE sqle_node_tcpip
{
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ+1];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-TCPIP.
    05 HOSTNAME               PIC X(255).
    05 FILLER                 PIC X.
    05 SERVICE-NAME           PIC X(14).
    05 FILLER                 PIC X.
*
```

Related reference:

- "sqlectnd - Catalog Node" on page 322
- "SQLE-NODE-STRUCT" on page 499

SQLE-REG-NWBINDERY

This structure is used to register (using the sqleregs API) or deregister (using the sqledreg API) the DB2 server on the bindery on the NetWare file server.

Table 41. Fields in the SQLE-REG-NWBINDERY Structure

Field Name	Data Type	Description
UID	CHAR(48)	User ID used to log into the NetWare file server.
PSWD	CHAR(128)	Password used to validate the user ID.

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-REG-NWBINDERY */
/* ... */
SQL_STRUCTURE sqle_reg_nwbindery
{
  char                    uid[SQL_NW_UID_SZ+1];
  unsigned short          reserved_len_1;
  char                    pswd[SQL_NW_PSWD_SZ+1];
  unsigned short          reserved_len_2;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-REG-NWBINDERY.
    05 SQL-UID                 PIC X(48).
    05 FILLER                  PIC X.
    05 FILLER                  PIC X(1).
    05 SQL-UID-LEN             PIC 9(4) COMP-5.
    05 SQL-PSWD                PIC X(128).
    05 FILLER                  PIC X.
    05 FILLER                  PIC X(1).
    05 SQL-PSWD-LEN            PIC 9(4) COMP-5.
*
```

Related reference:

- “sqleregs - Register” on page 380
- “sqledreg - Deregister” on page 338

SQLEDBTERRITORYINFO

This structure is used to provide code set and territory options to the sqlecrea API.

Table 42. Fields in the SQLEDBTERRITORYINFO Structure

Field Name	Data Type	Description
SQLDBCODESET	CHAR(9)	Database code set.
SQLDBLOCALE	CHAR(5)	Database territory.

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDBTERRITORYINFO */
/* ... */
SQL_STRUCTURE sqledbterritoryinfo
{
  char                   sqldbcodeset[SQL_CODESET_LEN + 1];
  char                   sqldblocale[SQL_LOCALE_LEN + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDBTERRITORYINFO.
    05 SQLDBCODESET          PIC X(9).
    05 FILLER                PIC X.
    05 SQLDBLOCALE           PIC X(5).
    05 FILLER                PIC X.
*
```

Related reference:

- "sqlecrea - Create Database" on page 314

SQLEDBDESC

The Database Description Block (SQLEDBDESC) structure can be used during a call to the sqlecrea API to specify permanent values for database attributes. These attributes include database comment, collating sequences, and table space definitions.

Table 43. Fields in the SQLEDBDESC Structure

Field Name	Data Type	Description
SQLDBDID	CHAR(8)	A structure identifier and "eye-catcher" for storage dumps. It is a string of eight bytes that must be initialized with the value of `SQLE_DBDESC_2` (defined in `sqlenv`). The contents of this field are validated for version control.
SQLDBCCP	INTEGER	The code page of the database comment. This value is no longer used by the database manager.
SQLDBCSS	INTEGER	A value indicating the source of the database collating sequence. See below for values. **Note:** To specify the IDENTITY collating sequence when creating a database, specify SQL_CS_NONE (which implements a binary collating sequence).
SQLDBUDC	CHAR(256)	The *n*th byte of this field contains the sort weight of the code point whose underlying decimal representation is *n* in the code page of the database. If SQLDBCSS is not equal to `SQL_CS_USER`, this field is ignored.
SQLDBCMT	CHAR(30)	The comment for the database.

Table 43. Fields in the SQLEDBDESC Structure (continued)

Field Name	Data Type	Description
SQLDBSGP	INTEGER	Reserved field. No longer used.
SQLDBNSG	SHORT	A value which indicates the number of file segments to be created in the database. The minimum value for this field is 1 and the maximum value for this field is 256. If a value of -1 is supplied, this field will default to 1. **Note:** SQLDBNSG set to zero produces a default for Version 1 compatibility.
SQLTSEXT	INTEGER	A value, in 4KB pages, which indicates the default extent size for each table space in the database. The minimum value for this field is 2 and the maximum value for this field is 256. If a value of -1 is supplied, this field will default to 32.
SQLCATTS	Pointer	A pointer to a table space description control block, SQLETSDESC, which defines the catalog table space. If null, a default catalog table space based on the values of SQLTSEXT and SQLDBNSG will be created.
SQLUSRTS	Pointer	A pointer to a table space description control block, SQLETSDESC, which defines the user table space. If null, a default user table space based on the values of SQLTSEXT and SQLDBNSG will be created.
SQLTMPTS	Pointer	A pointer to a table space description control block, SQLETSDESC, which defines the system temporary table space. If null, a default system temporary table space based on the values of SQLTSEXT and SQLDBNSG will be created.

The Tablespace Description Block structure (SQLETSDESC) is used to specify the attributes of any of the three initial table spaces.

Table 44. Fields in the SQLETSDESC Structure

Field Name	Data Type	Description
SQLTSDID	CHAR(8)	A structure identifier and "eye-catcher" for storage dumps. It is a string of eight bytes that must be initialized with the value of `SQLE_DBTSDESC_1` (defined in `sqlenv`). The contents of this field are validated for version control.
SQLEXTNT	INTEGER	Table space extent size, in 4 KB pages. If a value of -1 is supplied, this field will default to the current value of the *dft_extent_sz* configuration parameter.
SQLPRFTC	INTEGER	Table space prefetch size, in 4 KB pages. If a value of -1 is supplied, this field will default to the current value of the *dft_prefetch_sz* configuration parameter.
SQLPOVHD	DOUBLE	Table space I/O overhead, in milliseconds. If a value of -1 is supplied, this field will default to an internal database manager value (currently 24.1 ms) that could change with future releases.

Table 44. Fields in the SQLETSDESC Structure (continued)

Field Name	Data Type	Description
SQLTRFRT	DOUBLE	Table space I/O transfer rate, in milliseconds. If a value of -1 is supplied, this field will default to an internal database manager value (currently 0.9 ms) that could change with future releases.
SQLTSTYP	CHAR(1)	Indicates whether the table space is system-managed or database-managed. See below for values.
SQLCCNT	SMALLINT	Number of containers being assigned to the table space. Indicates how many SQLCTYPE/SQLCSIZE/SQLCLEN/SQLCONTR values follow.
CONTAINR	Array	An array of *sqlccnt SQLETSCDESC* structures.

Table 45. Fields in the SQLETSCDESC Structure

Field Name	Data Type	Description
SQLCTYPE	CHAR(1)	Identifies the type of this container. See below for values.
SQLCSIZE	INTEGER	Size of the container identified in *SQLCONTR*, specified in 4KB pages. Valid only when *SQLTSTYP* is set to `SQL_TBS_TYP_DMS`.
SQLCLEN	SMALLINT	Length of following *SQLCONTR* value.
SQLCONTR	CHAR(256)	Container string.

Valid values for *SQLDBCSS* (defined in `sqlenv`) are:

SQL_CS_SYSTEM
Collating sequence from system.

SQL_CS_USER
Collating sequence from user.

SQL_CS_NONE
None.

SQLE_CS_COMPATABILITY
Use pre-Version 5 collating sequence.

SQL_CS_SYSTEM_NLSCHAR
Collating sequence from system using the NLS version of compare routines for character types.

SQL_CS_USER_NLSCHAR
Collating sequence from user using the NLS version of compare routines for character types.

SQL_CS_IDENTITY_16BIT
A Unicode database can be created with the

SQL_CS_IDENTITY_16BIT collation option. SQL_CS_DENTITY_16BIT differs from the default SQL_CS_NONE collation option in that the CHAR, VARCHAR, LONG VARCHAR, and CLOB data in the Unicode database will be collated using the CESU-8 binary order instead of the UTF-8 binary order. CESU-8 is Compatibility Encoding Scheme for UTF-16: 8-Bit, and as of this writing, its specification is contained in the Draft Unicode Technical Report #26 available at the Unicode Technical Consortium web site (www.unicode.org). CESU-8 is binary identical to UTF-8 except for the Unicode supplementary characters, that is, those characters that are defined outside the 16-bit Basic Multilingual Plane (BMP or Plane 0). In UTF-8 encoding, a supplementary character is represented by one 4-byte sequence, but the same character in CESU-8 requires two 3-byte sequences. In a Unicode database, CHAR, VARCHAR, LONG VARCHAR, and CLOB data are stored in UTF-8, and GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, and DBCLOB data are stored in UCS-2. For SQL_CS_NONE collation, non-supplementary characters in UTF-8 and UCS-2 have identical binary collation, but supplementary characters in UTF-8 collate differently from the same characters in UCS-2. SQL_CS_IDENTITY_16BIT ensures all characters, supplementary and non-supplementary, in a DB2 Unicode databases have the same binary collation.

Valid values for *SQLTSTYP* (defined in `sqlenv`) are:

SQL_TBS_TYP_SMS
System managed

SQL_TBS_TYP_DMS
Database managed.

Valid values for *SQLCTYPE* (defined in `sqlenv`) are:

SQL_TBSC_TYP_DEV
Device. Valid only when *SQLTSTYP* = `SQL_TBS_TYP_DMS`.

SQL_TBSC_TYP_FILE
File. Valid only when *SQLTSTYP* = `SQL_TBS_TYP_DMS`.

SQL_TBSC_TYP_PATH
Path (directory). Valid only when *SQLTSTYP* = `SQL_TBS_TYP_SMS`.

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDBDESC */
/* ... */
SQL_STRUCTURE sqledbdesc
{
  _SQLOLDCHAR    sqldbdid[8];
  sqlint32       sqldbccp;
  sqlint32       sqldbcss;
  unsigned char  sqldbudc[SQL_CS_SZ];
  _SQLOLDCHAR    sqldbcmt[SQL_CMT_SZ+1];
  _SQLOLDCHAR    pad[1];
  sqluint32      sqldbsgp;
  short          sqldbnsg;
  char           pad2[2];
  sqlint32       sqltsext;
  struct SQLETSDESC *sqlcatts;
  struct SQLETSDESC *sqlusrts;
  struct SQLETSDESC *sqltmpts;
};
/* ... */
```

```
/* File: sqlenv.h */
/* Structure: SQLETSDESC */
/* ... */
SQL_STRUCTURE SQLETSDESC
{
  char           sqltsdid[8];
  sqlint32       sqlextnt;
  sqlint32       sqlprftc;
  double         sqlpovhd;
  double         sqltrfrt;
  char           sqltstyp;
  char           pad1;
  short          sqlccnt;
  struct SQLETSCDESC containr[1];
};
/* ... */
```

```
/* File: sqlenv.h */
/* Structure: SQLETSCDESC */
/* ... */
SQL_STRUCTURE SQLETSCDESC
{
  char           sqlctype;
  char           pad1[3];
  sqlint32       sqlcsize;
  short          sqlclen;
  char           sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
  char           pad2[2];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDBDESC.
    05 SQLDBDID               PIC X(8).
    05 SQLDBCCP               PIC S9(9) COMP-5.
    05 SQLDBCSS               PIC S9(9) COMP-5.
    05 SQLDBUDC               PIC X(256).
    05 SQLDBCMT               PIC X(30).
    05 FILLER                 PIC X.
    05 SQL-PAD                PIC X(1).
    05 SQLDBSGP               PIC 9(9) COMP-5.
    05 SQLDBNSG               PIC S9(4) COMP-5.
    05 SQL-PAD2               PIC X(2).
    05 SQLTSEXT               PIC S9(9) COMP-5.
    05 SQLCATTS               USAGE IS POINTER.
    05 SQLUSRTS               USAGE IS POINTER.
    05 SQLTMPTS               USAGE IS POINTER.
*
* File: sqletsd.cbl
01 SQLETSDESC.
    05 SQLTSDID               PIC X(8).
    05 SQLEXTNT               PIC S9(9) COMP-5.
    05 SQLPRFTC               PIC S9(9) COMP-5.
    05 SQLPOVHD               USAGE COMP-2.
    05 SQLTRFRT               USAGE COMP-2.
    05 SQLTSTYP               PIC X.
    05 SQL-PAD1               PIC X.
    05 SQLCCNT                PIC S9(4) COMP-5.
    05 SQL-CONTAINR OCCURS 001 TIMES.
        10 SQLCTYPE           PIC X.
        10 SQL-PAD1           PIC X(3).
        10 SQLCSIZE           PIC S9(9) COMP-5.
        10 SQLCLEN            PIC S9(4) COMP-5.
        10 SQLCONTR           PIC X(256).
        10 SQL-PAD2           PIC X(2).
*
* File: sqlenv.cbl
01 SQLETSCDESC.
    05 SQLCTYPE               PIC X.
    05 SQL-PAD1               PIC X(3).
    05 SQLCSIZE               PIC S9(9) COMP-5.
    05 SQLCLEN                PIC S9(4) COMP-5.
    05 SQLCONTR               PIC X(256).
    05 SQL-PAD2               PIC X(2).
*
```

Related reference:

- “sqlecrea - Create Database” on page 314

SQLEDINFO

This structure is used to return information after a call to the sqledgne API. It is shared by both the system database directory and the local database directory.

Table 46. Fields in the SQLEDINFO Structure

Field Name	Data Type	Description
ALIAS	CHAR(8)	An alternate database name.
DBNAME	CHAR(8)	The name of the database.
DRIVE	CHAR(215)	The local database directory path name where the database resides. This field is returned only if the system database directory is opened for scan. **Note:** On Windows NT, it is CHAR(12).
INTNAME	CHAR(8)	A token identifying the database subdirectory. This field is returned only if the local database directory is opened for scan.
NODENAME	CHAR(8)	The name of the node where the database is located. This field is returned only if the cataloged database is a remote database.
DBTYPE	CHAR(20)	Database manager release information.
COMMENT	CHAR(30)	The comment associated with the database.
COM_CODEPAGE	SMALLINT	The code page of the comment. Not used.
TYPE	CHAR(1)	Entry type. See below for values.
AUTHENTICATION	SMALLINT	Authentication type. See below for values.
GLBDBNAME	CHAR(255)	The global name of the target database in the global (DCE) directory, if the entry is of type `SQL_DCE`.
DCEPRINCIPAL	CHAR(1024)	The principal name if the authentication is of type DCE or KERBEROS.
CAT_NODENUM	SHORT	Catalog node number.
NODENUM	SHORT	Node number.
Note: Both system and local database directory use the same structure, but only certain fields are valid for each. Each character field returned is blank filled up to the length of the field.		

Valid values for *TYPE* (defined in `sqlenv`) are:

SQL_INDIRECT
Database created by the current instance (as defined by the value of the **DB2INSTANCE** environment variable).

SQL_REMOTE
Database resides at a different instance.

SQL_HOME
Database resides on this volume (always HOME in local database directory).

SQL_DCE
Database resides in DCE directories.

Valid values for *AUTHENTICATION* (defined in `sqlenv`) are:

SQL_AUTHENTICATION_SERVER
Authentication of the user name and password takes place at the server.

SQL_AUTHENTICATION_CLIENT
Authentication of the user name and password takes place at the client.

SQL_AUTHENTICATION_DCS
Used for DB2 Connect.

SQL_AUTHENTICATION_DCE
Authentication takes place using DCE Security Services.

SQL_AUTHENTICATION_KERBEROS
Authentication takes place using Kerberos Security Mechanism.

SQL_AUTHENTICATION_NOT_SPECIFIED
DB2 no longer requires authentication to be kept in the database directory. Specify this value when connecting to anything other than a down-level (DB2 V2 or less) server.

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDINFO */
/* ... */
SQL_STRUCTURE sqledinfo
{
  _SQLOLDCHAR    alias[SQL_ALIAS_SZ];
  _SQLOLDCHAR    dbname[SQL_DBNAME_SZ];
  _SQLOLDCHAR    drive[SQL_DRIVE_SZ];
  _SQLOLDCHAR    intname[SQL_INAME_SZ];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    dbtype[SQL_DBTYP_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  short          com_codepage;
  _SQLOLDCHAR    type;
  unsigned short authentication;
  char           glbdbname[SQL_DIR_NAME_SZ];
  _SQLOLDCHAR    dceprincipal[SQL_DCEPRIN_SZ];
  short          cat_nodenum;
  short          nodenum;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDINFO.
    05 SQL-ALIAS              PIC X(8).
    05 SQL-DBNAME             PIC X(8).
    05 SQL-DRIVE              PIC X(215).
    05 SQL-INTNAME            PIC X(8).
    05 SQL-NODENAME           PIC X(8).
    05 SQL-DBTYPE             PIC X(20).
    05 SQL-COMMENT            PIC X(30).
    05 FILLER                 PIC X(1).
    05 SQL-COM-CODEPAGE       PIC S9(4) COMP-5.
    05 SQL-TYPE               PIC X.
    05 FILLER                 PIC X(1).
    05 SQL-AUTHENTICATION     PIC 9(4) COMP-5.
    05 SQL-GLBDBNAME          PIC X(255).
    05 SQL-DCEPRINCIPAL       PIC X(1024).
    05 FILLER                 PIC X(1).
    05 SQL-CAT-NODENUM        PIC S9(4) COMP-5.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
*
```

Related reference:

- "sqledgne - Get Next Database Directory Entry" on page 331

SQLENINFO

This structure returns information after a call to the sqlengne API.

Table 47. Fields in the SQLENINFO Structure

Field Name	Data Type	Description
NODENAME	CHAR(8)	Used for the NetBIOS protocol; the *nname* of the node where the database is located (valid in system directory only).
LOCAL_LU	CHAR(8)	Used for the APPN protocol; local logical unit.
PARTNER_LU	CHAR(8)	Used for the APPN protocol; partner logical unit.
MODE	CHAR(8)	Used for the APPN protocol; transmission service mode.
COMMENT	CHAR(30)	The comment associated with the node.
COM_CODEPAGE	SMALLINT	The code page of the comment. This field is no longer used by the database manager.
ADAPTER	SMALLINT	Used for the NetBIOS protocol; the local network adapter.
NETWORKID	CHAR(8)	Used for the APPN protocol; network ID.
PROTOCOL	CHAR(1)	Communications protocol.
SYM_DEST_NAME	CHAR(8)	Used for the APPC protocol; the symbolic destination name.
SECURITY_TYPE	SMALLINT	Used for the APPC protocol; the security type. See below for values.
HOSTNAME	CHAR(255)	Used for the TCP/IP protocol; the name of the TCP/IP host on which the DB2 server instance resides.
SERVICE_NAME	CHAR(14)	Used for the TCP/IP protocol; the TCP/IP service name or associated port number of the DB2 server instance.
FILESERVER	CHAR(48)	Used for the IPX/SPX protocol; the name of the NetWare file server where the DB2 server instance is registered.
OBJECTNAME	CHAR(48)	The database manager server instance is represented as the object, *objectname*, on the NetWare file server. The server's IPX/SPX internetwork address is stored and retrieved from this object.
INSTANCE_NAME	CHAR(8)	Used for the local and NPIPE protocols; the name of the server instance.
COMPUTERNAME	CHAR(15)	Used by the NPIPE protocol; the server node's computer name.
SYSTEM_NAME	CHAR(21)	The DB2 system name of the remote server.
REMOTE_INSTNAME	CHAR(8)	The name of the DB2 server instance.

Table 47. Fields in the SQLENINFO Structure (continued)

Field Name	Data Type	Description
CATALOG_NODE_TYPE	CHAR	Catalog node type.
OS_TYPE	UNSIGNED SHORT	Identifies the operating system of the server.
Note: Each character field returned is blank filled up to the length of the field.		

Valid values for *SECURITY_TYPE* (defined in `sqlenv`) are:

SQL_CPIC_SECURITY_NONE

SQL_CPIC_SECURITY_SAME

SQL_CPIC_SECURITY_PROGRAM

Language syntax:

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLENINFO */
/* ... */
SQL_STRUCTURE sqleninfo
{
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ];
  _SQLOLDCHAR    mode[SQL_MODE_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  unsigned short com_codepage;
  unsigned short adapter;
  _SQLOLDCHAR    networkid[SQL_NETID_SZ];
  _SQLOLDCHAR    protocol;
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ];
  unsigned short security_type;
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ];
  char           fileserver[SQL_FILESERVER_SZ];
  char           objectname[SQL_OBJECTNAME_SZ];
  char           instance_name[SQL_INSTNAME_SZ];
  char           computername[SQL_COMPUTERNAME_SZ];
  char           system_name[SQL_SYSTEM_NAME_SZ];
  char           remote_instname[SQL_REMOTE_INSTNAME_SZ];
  _SQLOLDCHAR    catalog_node_type;
  unsigned short os_type;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLENINFO.
    05 SQL-NODE-NAME          PIC X(8).
    05 SQL-LOCAL-LU           PIC X(8).
    05 SQL-PARTNER-LU         PIC X(8).
    05 SQL-MODE               PIC X(8).
    05 SQL-COMMENT            PIC X(30).
    05 SQL-COM-CODEPAGE       PIC 9(4) COMP-5.
    05 SQL-ADAPTER            PIC 9(4) COMP-5.
    05 SQL-NETWORKID          PIC X(8).
    05 SQL-PROTOCOL           PIC X.
    05 SQL-SYM-DEST-NAME      PIC X(8).
    05 FILLER                 PIC X(1).
    05 SQL-SECURITY-TYPE      PIC 9(4) COMP-5.
    05 SQL-HOSTNAME           PIC X(255).
    05 SQL-SERVICE-NAME       PIC X(14).
    05 SQL-FILESERVER         PIC X(48).
    05 SQL-OBJECTNAME         PIC X(48).
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 SQL-COMPUTERNAME       PIC X(15).
    05 SQL-SYSTEM-NAME        PIC X(21).
    05 SQL-REMOTE-INSTNAME    PIC X(8).
    05 SQL-CATALOG-NODE-TYPE  PIC X.
    05 SQL-OS-TYPE            PIC 9(4) COMP-5.
*
```

Related reference:

- "sqlengne - Get Next Node Directory Entry" on page 371

SQLFUPD

This structure passes information about database configuration files and the database manager configuration file.

Table 48. Fields in the SQLFUPD Structure

Field Name	Data Type	Description
TOKEN	UINT16	Specifies the configuration value to return or update.
PTRVALUE	Pointer	A pointer to an application allocated buffer that holds the data specified by *TOKEN*.

Valid data types for the *token* element are:

Uint16	Unsigned 2-byte integer
Sint16	Signed 2-byte integer
Uint32	Unsigned 4-byte integer
Sint32	Signed 4-byte integer

float	4-byte floating-point decimal
char(*n*)	String of length *n* (not including null termination).

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLFUPD */
/* ... */
SQL_STRUCTURE sqlfupd
{
  unsigned short  token;
  char            *ptrvalue;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-FUPD.
    05 SQL-TOKEN              PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-VALUE-PTR          USAGE IS POINTER.
*
```

SQLM-COLLECTED

This structure is used to return information after a call to the Database System Monitor APIs. It will only be filled in for snapshot requests made at the SQLM_DBMON_VERSION5_2 level and lower.

Table 49. Fields in the SQLM-COLLECTED Structure

Field Name	Data Type	Description
SIZE	sqluint32	The size of the structure.
DB2	sqluint32	Obsolete.
DATABASES	sqluint32	Obsolete.
TABLE_DATABASES	sqluint32	Obsolete.
LOCK_DATABASES	sqluint32	Obsolete.
APPLICATIONS	sqluint32	Obsolete.
APPLINFOS	sqluint32	Obsolete.
DCS_APPLINFOS	sqluint32	Obsolete.
SERVER_DB2_TYPE	sqluint32	The database manager server type (defined in `sqlutil.h`).
TIME_STAMP	TIMESTAMP	Time that the snapshot was taken.
GROUP_STATES	OBJECT SQLM_ RECORDING_ GROUP	Current state of the monitor switch.

Table 49. Fields in the SQLM-COLLECTED Structure (continued)

Field Name	Data Type	Description
SERVER_PRDID	CHAR(20)	Product name and version number of the database manager on the server.
SERVER_NNAME	CHAR(20)	Configuration node name of the server.
SERVER_ INSTANCE_NAME	CHAR(20)	Instance name of the database manager.
RESERVED	CHAR(22)	Reserved for future use.
NODE_NUMBER	UNSIGNED SHORT	Number of the node sending data.
TIME_ZONE_DISP	sqlint32	The difference (in seconds) between GMT and local time.
NUM_TOP_LEVEL_ STRUCTS	sqluint32	The total number of high-level structures returned in the snapshot output buffer. A high-level structure can be composed of several lower-level data structures. This counter replaces the individual counters (such as *table_databases*) for each high-level structure, which are now obsolete.
TABLESPACE_ DATABASES	sqluint32	Obsolete.
SERVER_VERSION	sqluint32	The version of the server returning the data.

Language syntax:

C Structure

```
/* File: sqlmon.h */
/* Structure: SQLM-COLLECTED */
/* ... */
typedef struct sqlm_collected
{
  sqluint32      size;
  sqluint32      db2;
  sqluint32      databases;
  sqluint32      table_databases;
  sqluint32      lock_databases;
  sqluint32      applications;
  sqluint32      applinfos;
  sqluint32      dcs_applinfos;
  sqluint32      server_db2_type;
  sqlm_timestamp time_stamp;
  sqlm_recording_group group_states[SQLM_NUM_GROUPS];
  _SQLOLDCHAR    server_prdid[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_nname[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_instance_name[SQLM_IDENT_SZ];
  _SQLOLDCHAR    reserved[22];
  unsigned short node_number;
  long           time_zone_disp;
  sqluint32      num_top_level_structs;
  sqluint32      tablespace_databases;
  sqluint32      server_version;
}sqlm_collected;
/* ... */
```

COBOL Structure

```
* File: sqlmonct.cbl
01 SQLM-COLLECTED.
    05 SQLM-SIZE               PIC 9(9) COMP-5.
    05 DB2                     PIC 9(9) COMP-5.
    05 DATABASES               PIC 9(9) COMP-5.
    05 TABLE-DATABASES         PIC 9(9) COMP-5.
    05 LOCK-DATABASES          PIC 9(9) COMP-5.
    05 APPLICATIONS            PIC 9(9) COMP-5.
    05 APPLINFOS               PIC 9(9) COMP-5.
    05 DCS-APPLINFOS           PIC 9(9) COMP-5.
    05 SERVER-DB2-TYPE         PIC 9(9) COMP-5.
    05 TIME-STAMP.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
    05 GROUP-STATES OCCURS 6.
        10 INPUT-STATE          PIC 9(9) COMP-5.
        10 OUTPUT-STATE         PIC 9(9) COMP-5.
        10 START-TIME.
    05 SERVER-PRDID            PIC X(20).
    05 SERVER-NNAME            PIC X(20).
    05 SERVER-INSTANCE-NAME    PIC X(20).
    05 RESERVED                PIC X(32).
    05 TABLESPACE-DATABASES    PIC 9(9) COMP-5.
    05 SERVER-VERSION          PIC 9(9) COMP-5.
*
```

SQLM-RECORDING-GROUP

This structure is used to return information after a call to the Database System Monitor APIs.

Table 50. Fields in the SQLM-RECORDING-GROUP Structure

Field Name	Data Type	Description
INPUT_STATE	INTEGER	Required state for the specific monitor group.
OUTPUT_STATE	INTEGER	Returned information on the state of the specific monitor switch.
START_TIME	Structure	Time stamp when the monitoring group switch was turned on.

Table 51. Fields in the SQLM-TIMESTAMP Structure

Field Name	Data Type	Description
SECONDS	INTEGER	The date and time, expressed as the number of seconds since January 1, 1970 (GMT).

Table 51. Fields in the SQLM-TIMESTAMP Structure (continued)

Field Name	Data Type	Description
MICROSEC	INTEGER	The number of elapsed microseconds in the current second.

For both *input_state* and *output_state*, a particular monitor switch is identified by its index in the array passed to the db2MonitorSwitches API. The constants that map the indexes to the switches are called `SQLM_`*XXXX*`_SW`, where *XXXX* is the name of the monitor group. The constants are defined in `sqlmon.h`.

Language syntax:

C Structure

```
/* File: sqlmon.h */
/* Structure: SQLM-RECORDING-GROUP */
/* ... */
typedef struct sqlm_recording_group
{
  sqluint32      input_state;
  sqluint32      output_state;
  sqlm_timestamp start_time;
}sqlm_recording_group;
/* ... */
```

```
/* File: sqlmon.h */
/* Structure: SQLM-TIMESTAMP */
/* ... */
typedef struct sqlm_timestamp
{
  sqluint32  seconds;
  sqluint32  microsec;
}sqlm_timestamp;
/* ... */
```

COBOL Structure

```
* File: sqlmonct.cbl
01 SQLM-RECORDING-GROUP OCCURS 6 TIMES.
    05 INPUT-STATE             PIC 9(9) COMP-5.
    05 OUTPUT-STATE            PIC 9(9) COMP-5.
    05 START-TIME.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
*
```

```
* File: sqlmonct.cbl
01 SQLM-TIMESTAMP.
    05 SECONDS                 PIC 9(9) COMP-5.
    05 MICROSEC                PIC 9(9) COMP-5.
*
```

Related reference:

- "db2MonitorSwitches - Get/Update Monitor Switches" on page 177

SQLMA

The SQL monitor area (SQLMA) structure is used to send database monitor snapshot requests to the database manager. It is also used to estimate the size (in bytes) of the snapshot output.

Table 52. Fields in the SQLMA Structure

Field Name	Data Type	Description
OBJ_NUM	INTEGER	Number of objects to be monitored.
OBJ_VAR	Array	An array of *sqlm_obj_struct* structures containing descriptions of objects to be monitored. The length of the array is determined by *OBJ_NUM*.

Table 53. Fields in the SQLM-OBJ-STRUCT Structure

Field Name	Data Type	Description
AGENT_ID	INTEGER	The application handle of the application to be monitored. Specified only if *OBJ_TYPE* requires an *agent_id* (application handle).
OBJ_TYPE	INTEGER	The type of object to be monitored.
OBJECT	CHAR(36)	The name of the object to be monitored. Specified only if *OBJ_TYPE* requires a name, such as *appl_id*, or a database alias.

Valid values for *OBJ_TYPE* (defined in `sqlmon`) are:

SQLMA_DB2
DB2 related information

SQLMA_DBASE
Database related information

SQLMA_APPL
Application information organized by the application ID

SQLMA_AGENT_ID
Application information organized by the agent ID

SQLMA_DBASE_TABLES
Table information for a database

SQLMA_DBASE_APPLS
Application information for a database

SQLMA_DBASE_APPLINFO
Summary application information for a database

SQLMA_DBASE_LOCKS
Locking information for a database

SQLMA_DBASE_ALL
Database information for all active databases in the database manager

SQLMA_APPL_ALL
Application information for all active applications in the database manager

SQLMA_APPLINFO_ALL
Summary application information for all active applications in the database manager

SQLMA_DCS_APPLINFO_ALL
Database Connection Services application information summary for all active applications in the database manager.

SQLMA_DYNAMIC_SQL
Get snapshot for dynamic SQL.

SQLMA_DCS_DBASE
Database Connection Services database level information.

SQLMA_DCS_DBASE_ALL
Database Connection Services database information for all active databases.

SQLMA_DCS_APPL_ALL
Database Connection Services application information for all connections.

SQLMA_DCS_APPL
Database Connection Services application information identified by application ID.

SQLMA_DCS_APPL_HANDLE
Database Connection Services application information identified by application handle.

SQLMA_DCS_DBASE_APPLS
Database Connection Services application information for all active connections to the database.

SQLMA_DBASE_TABLESPACES
Table space information for a database.

SQLMA_DBASE_REMOTE
Information for a DataJoiner database.

SQLMA_DBASE_REMOTE_ALL
Information for all DataJoiner databases.

SQLMA_DBASE_APPLS_REMOTE
Application information for a particular DataJoiner database.

SQLMA_APPLS_REMOTE_ALL
Application information for all DataJoiner databases.

Language syntax:

C Structure

```
/* File: sqlmon.h */
/* Structure: SQLMA */
/* ... */
typedef struct sqlma
{
  sqluint32 obj_num;
  sqlm_obj_struct obj_var[1];
}sqlma;
/* ... */
```

```
/* File: sqlmon.h */
/* Structure: SQLM-OBJ-STRUCT */
/* ... */
typedef struct sqlm_obj_struct
{
  sqluint32     agent_id;
  sqluint32     obj_type;
  _SQLOLDCHAR   object[SQLM_OBJECT_SZ];
}sqlm_obj_struct;
/* ... */
```

COBOL Structure

```
* File: sqlmonct.cbl
01 SQLMA.
    05 OBJ-NUM                 PIC 9(9) COMP-5.
    05 OBJ-VAR OCCURS 0 TO 100 TIMES DEPENDING ON OBJ-NUM.
        10 AGENT-ID             PIC 9(9) COMP-5.
        10 OBJ-TYPE             PIC 9(9) COMP-5.
        10 OBJECT               PIC X(36).
*
```

SQLOPT

This structure is used to pass bind options to the sqlabndx API, precompile options to the sqlaprep API, and rebind options to the sqlarbnd API.

Table 54. Fields in the SQLOPT Structure

Field Name	Data Type	Description
HEADER	Structure	An *sqloptheader* structure.

Table 54. Fields in the SQLOPT Structure (continued)

Field Name	Data Type	Description
OPTION	Array	An array of *sqloptions* structures. The number of elements in this array is determined by the value of the *allocated* field of the *header*.

Table 55. Fields in the SQLOPTHEADER Structure

Field Name	Data Type	Description
ALLOCATED	INTEGER	Number of elements in the *option* array of the *sqlopt* structure.
USED	INTEGER	Number of elements in the *option* array of the *sqlopt* structure actually used. This is the number of option pairs (*TYPE* and *VAL*) supplied.

Table 56. Fields in the SQLOPTIONS Structure

Field Name	Data Type	Description
TYPE	INTEGER	Bind/precompile/rebind option type.
VAL	INTEGER	Bind/precompile/rebind option value.
Note: The *TYPE* and *VAL* fields are repeated for each bind/precompile/rebind option specified.		

Language syntax:

C Structure

```
/* File: sql.h */
/* Structure: SQLOPT */
/* ... */
SQL_STRUCTURE sqlopt
{
  SQL_STRUCTURE sqloptheader header;
  SQL_STRUCTURE sqloptions   option[1];
};
/* ... */
```

```
/* File: sql.h */
/* Structure: SQLOPTHEADER */
/* ... */
SQL_STRUCTURE sqloptheader
{
  sqluint32 allocated;
  sqluint32 used;
};
/* ... */
```

```
/* File: sql.h */
/* Structure: SQLOPTIONS */
/* ... */
SQL_STRUCTURE sqloptions
{
  sqluint32 type;
  sqluint32 val;
};
/* ... */
```

COBOL Structure

```
* File: sql.cbl
01 SQLOPT.
    05 SQLOPTHEADER.
        10 ALLOCATED   PIC 9(9) COMP-5.
        10 USED        PIC 9(9) COMP-5.
    05 SQLOPTIONS OCCURS 1 TO 50 DEPENDING ON ALLOCATED.
        10 SQLOPT-TYPE        PIC 9(9) COMP-5.
        10 SQLOPT-VAL         PIC 9(9) COMP-5.
        10 SQLOPT-VAL-PTR     REDEFINES SQLOPT-VAL
*
```

Related reference:

- “sqlabndx - Bind” on page 253
- “sqlaprep - Precompile Program” on page 259
- “sqlarbnd - Rebind” on page 262

SQLU-LSN

This union, used by the db2ReadLog API, contains the definition of the log sequence number. A log sequence number (LSN) represents a relative byte address within the database log. All log records are identified by this number. It represents the log record’s byte offset from the beginning of the database log.

Table 57. Fields in the SQLU-LSN Union

Field Name	Data Type	Description
lsnChar	Array of UNSIGNED CHAR	Specifies the 6-member character array log sequence number.
lsnWord	Array of UNSIGNED SHORT	Specifies the 3-member short array log sequence number.

Language syntax:

C Structure

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

Related reference:

- "db2ReadLog - Asynchronous Read Log" on page 185

SQLU-MEDIA-LIST

This structure is used to pass information to the db2Load API.

Table 58. Fields in the SQLU-MEDIA-LIST Structure

Field Name	Data Type	Description
MEDIA_TYPE	CHAR(1)	A character indicating media type.
SESSIONS	INTEGER	Indicates the number of elements in the array pointed to by the *target* field of this structure.
TARGET	Union	This field is a pointer to one of four types of structures. The type of structure pointed to is determined by the value of the *media_type* field. For more information on what to provide in this field, see the appropriate API.

Table 59. Fields in the SQLU-MEDIA-LIST-TARGETS Structure

Field Name	Data Type	Description
MEDIA	Pointer	A pointer to an *sqlu_media_entry* structure.
VENDOR	Pointer	A pointer to an *sqlu_vendor* structure.
LOCATION	Pointer	A pointer to an *sqlu_location_entry* structure.
PSTATEMENT	Pointer	A pointer to an *sqlu_statement_entry* structure.

Table 60. Fields in the SQLU-MEDIA-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the *media_entry* field. For languages other than C.
MEDIA_ENTRY	CHAR(215)	Path for a backup image used by the backup and restore utilities.

Table 61. Fields in the SQLU-VENDOR Structure

Field Name	Data Type	Description
RESERVE_LEN1	INTEGER	Length of the *shr_lib* field. For languages other than C.
SHR_LIB	CHAR(255)	Name of a shared library supplied by vendors for storing or retrieving data.
RESERVE_LEN2	INTEGER	Length of the *filename* field. For languages other than C.
FILENAME	CHAR(255)	File name to identify the load input source when using a shared library.

Table 62. Fields in the SQLU-LOCATION-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the *location_entry* field. For languages other than C.
LOCATION_ENTRY	CHAR(256)	Name of input data files for the load utility.

Table 63. Fields in the SQLU-STATEMENT-ENTRY Structure

Field Name	Data Type	Description
LENGTH	INTEGER	Length of the *data* field.
PDATA	Pointer	Pointer to the SQL query.

Valid values for *MEDIA_TYPE* (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices (tapes, disks, or diskettes)

SQLU_SERVER_LOCATION
Server devices (tapes, disks, or diskettes; load only). Can be specified only for the *piSourceList* parameter.

SQLU_CLIENT_LOCATION
Client devices (files or named pipes; load only). Can be specified only for the *piSourceList* parameter.

SQLU_SQL_STMT
SQL query (load only). Can be specified only for the *piSourceList* parameter.

SQLU_TSM_MEDIA
TSM

SQLU_OTHER_MEDIA
Vendor library

SQLU_USER_EXIT
User exit (OS/2 only)

SQLU_PIPE_MEDIA
Named pipe (for vendor APIs only)

SQLU_DISK_MEDIA
Disk (for vendor APIs only)

SQLU_DISKETTE_MEDIA
Diskette (for vendor APIs only)

SQLU_TAPE_MEDIA
Tape (for vendor APIs only).

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_media_list
{
  char           media_type;
  char           filler[3];
  sqlint32           sessions;
  union sqlu_media_list_targets target;
} sqlu_media_list;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST-TARGETS */
/* ... */
union sqlu_media_list_targets
{
  struct sqlu_media_entry       *media;
  struct sqlu_vendor            *vendor;
  struct sqlu_location_entry    *location;
  struct sqlu_statement_entry   *pStatement;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_media_entry
{
  sqluint32      reserve_len;
  char           media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-VENDOR */
/* ... */
typedef SQL_STRUCTURE sqlu_vendor
{
  sqluint32      reserve_len1;
  char           shr_lib[SQLU_SHR_LIB_LEN+1];
  sqluint32      reserve_len2;
  char           filename[SQLU_SHR_LIB_LEN+1];
} sqlu_vendor;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-LOCATION-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_location_entry
{
  sqluint32       reserve_len;
  char            location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_entry;
/* ... */
/* File: sqlutil.h */
/* Structure: SQLU-STATEMENT-ENTRY */
/* ... */
SQL_STRUCTURE sqlu_statement_entry
{
  sqluint32       length;
  char            *pEntry;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLU-MEDIA-LIST.
    05 SQL-MEDIA-TYPE          PIC X.
    05 SQL-FILLER              PIC X(3).
    05 SQL-SESSIONS            PIC S9(9) COMP-5.
    05 SQL-TARGET.
        10 SQL-MEDIA           USAGE IS POINTER.
        10 SQL-VENDOR          REDEFINES SQL-MEDIA
        10 SQL-LOCATION        REDEFINES SQL-MEDIA
        10 SQL-STATEMENT       REDEFINES SQL-MEDIA
        10 FILLER              REDEFINES SQL-MEDIA
*
* File: sqlutil.cbl
01 SQLU-MEDIA-ENTRY.
    05 SQL-MEDENT-LEN          PIC 9(9) COMP-5.
    05 SQL-MEDIA-ENTRY         PIC X(215).
    05 FILLER                  PIC X.
*
* File: sqlutil.cbl
01 SQLU-VENDOR.
    05 SQL-SHRLIB-LEN          PIC 9(9) COMP-5.
    05 SQL-SHR-LIB             PIC X(255).
    05 FILLER                  PIC X.
    05 SQL-FILENAME-LEN        PIC 9(9) COMP-5.
    05 SQL-FILENAME            PIC X(255).
    05 FILLER                  PIC X.
*
* File: sqlutil.cbl
01 SQLU-LOCATION-ENTRY.
    05 SQL-LOCATION-LEN        PIC 9(9) COMP-5.
    05 SQL-LOCATION-ENTRY      PIC X(255).
    05 FILLER                  PIC X.
*
```

```
* File: sqlutil.cbl
01 SQLU-STATEMENT-ENTRY.
    05 SQL-STATEMENT-LEN       PIC 9(9) COMP-5.
    05 SQL-STATEMENT-ENTRY     USAGE IS POINTER.
*
```

SQLU-RLOG-INFO

This structure contains information about the status of calls to the db2ReadLog, and the database log.

Table 64. Fields in the SQLU-RLOG-INFO Structure

Field Name	Data Type	Description
initialLSN	SQLU_LSN	Specifies the LSN value of the first log record that is written after the first database CONNECT statement is issued. For more information, see SQLU-LSN.
firstReadLSN	SQLU_LSN	Specifies the LSN value of the first log record read.
lastReadLSN	SQLU_LSN	Specifies the LSN value of the last log record read.
curActiveLSN	SQLU_LSN	Specifies the LSN value of the current (active) log.
logRecsWritten	sqluint32	Specifies the number of log records written to the buffer.
logBytesWritten	sqluint32	Specifies the number of bytes written to the buffer.

Language syntax:

C Structure

```
typedef SQL_STRUCTURE SQLU_RLOG_INFO
{
SQLU_LSN       initialLSN ;
SQLU_LSN       firstReadLSN ;
SQLU_LSN       lastReadLSN ;
SQLU_LSN       curActiveLSN ;
sqluint32      logRecsWritten ;
sqluint32      logBytesWritten ;
} SQLU_RLOG_INFO;
```

Related reference:

- “db2ReadLog - Asynchronous Read Log” on page 185
- “SQLU-LSN” on page 523

SQLUEXPT-OUT

This structure is used to pass information from sqluexpr.

Table 65. Fields in the SQLUEXPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of the structure.
ROWSEXPORTED	INTEGER	Number of records exported from the database into the target file.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-UEXPT-OUT */
/* ... */
SQL_STRUCTURE sqluexpt_out
{
  sqluint32      sizeOfStruct;
  sqluint32      rowsExported;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UEXPT-OUT.
    05 SQL-SIZE-OF-UEXPT-OUT  PIC 9(9) COMP-5 VALUE 8.
    05 SQL-ROWSEXPORTED       PIC 9(9) COMP-5 VALUE 0.
*
```

Related reference:

- "sqluexpr - Export" on page 408

SQLUIMPT-IN

This structure is used to pass information to sqluimpr.

Table 66. Fields in the SQLUIMPT-IN Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
COMMITCNT	INTEGER	The number of records to import before committing them to the database. A COMMIT is performed whenever *commitcnt* records are imported.

Table 66. Fields in the SQLUIMPT-IN Structure (continued)

Field Name	Data Type	Description
RESTARTCNT	INTEGER	The number of records to skip before starting to insert or update records. This parameter should be used if a previous attempt to import records fails after some records have been committed to the database. The specified value represents a starting point for the next import operation.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-IN */
/* ... */
SQL_STRUCTURE sqluimpt_in
{
  sqluint32       sizeOfStruct;
  sqluint32       commitcnt;
  sqluint32       restartcnt;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-IN.
    05 SQL-SIZE-OF-UIMPT-IN   PIC 9(9) COMP-5 VALUE 12.
    05 SQL-COMMITCNT          PIC 9(9) COMP-5 VALUE 0.
    05 SQL-RESTARTCNT         PIC 9(9) COMP-5 VALUE 0.
*
```

Related reference:

- “sqluimpr - Import” on page 424

SQLUIMPT-OUT

This structure is used to pass information from sqluimpr.

Table 67. Fields in the SQLUIMPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
ROWSREAD	INTEGER	Number of records read from the file during import.
ROWSSKIPPED	INTEGER	Number of records skipped before inserting or updating begins.

Table 67. Fields in the SQLUIMPT-OUT Structure (continued)

Field Name	Data Type	Description
ROWSINSERTED	INTEGER	Number of rows inserted into the target table.
ROWSUPDATED	INTEGER	Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).
ROWSREJECTED	INTEGER	Number of records that could not be imported.
ROWSCOMMITTED	INTEGER	The total number of processed records: the number of records inserted successfully and committed to the database, plus the number of skipped and rejected records.

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-OUT */
/* ... */
SQL_STRUCTURE sqluimpt_out
{
  sqluint32       sizeOfStruct;
  sqluint32       rowsRead;
  sqluint32       rowsSkipped;
  sqluint32       rowsInserted;
  sqluint32       rowsUpdated;
  sqluint32       rowsRejected;
  sqluint32       rowsCommitted;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-OUT.
    05 SQL-SIZE-OF-UIMPT-OUT  PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWSREAD           PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSSKIPPED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSINSERTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSUPDATED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSREJECTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSCOMMITTED      PIC 9(9) COMP-5 VALUE 0.
*
```

Related reference:

- "sqluimpr - Import" on page 424

SQLUPI

This structure is used to store partitioning information, such as the partitioning map and the partitioning key of a table.

Table 68. Fields in the SQLUPI Structure

Field Name	Data Type	Description
PMAPLEN	INTEGER	The length of the partitioning map in bytes. For a single-node table, the value is `sizeof(SQL_PDB_NODE_TYPE)`. For a mult-inode table, the value is `SQL_PDB_MAP_SIZE * sizeof(SQL_PDB_NODE_TYPE)`.
PMAP	SQL_PDB_NODE_TYPE	The partitioning map.
SQLD	INTEGER	The number of used SQLPARTKEY elements; that is, the number of key parts in a partitioning key.
SQLPARTKEY	Structure	The description of a partitioning column in a partitioning key. The maximum number of partitioning columns is SQL_MAX_NUM_PART_KEYS.

Table 69 shows the SQL data types and lengths for the SQLUPI data structure. The SQLTYPE column specifies the numeric value that represents the data type of an item.

Table 69. SQL Data Types and Lengths for the SQLUPI Structure

Data type	SQLTYPE (Nulls Not Allowed)	SQLTYPE (Nulls Allowed)	SQLLEN	AIX
Date	384	385	Ignored	Yes
Time	388	389	Ignored	Yes
Timestamp	392	393	Ignored	Yes
Variable-length character string	448	449	Length of the string	Yes
Fixed-length character string	452	453	Length of the string	Yes
Long character string	456	457	Ignored	No
Null-terminated character string	460	461	Length of the string	Yes
Floating point	480	481	Ignored	Yes
Decimal	484	485	Byte 1 = precision Byte 2 = scale	Yes
Large integer	496	497	Ignored	Yes
Small integer	500	501	Ignored	Yes

Table 69. SQL Data Types and Lengths for the SQLUPI Structure (continued)

Data type	SQLTYPE (Nulls Not Allowed)	SQLTYPE (Nulls Allowed)	SQLLEN	AIX
Variable-length graphic string	464	465	Length in double-byte characters	Yes
Fixed-length graphic string	468	469	Length in double-byte characters	Yes
Long graphic string	472	473	Ignored	No

Language syntax:

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUPI */
/* ... */
SQL_STRUCTURE sqlupi
{
  unsigned short  pmaplen;
  SQL_PDB_NODE_TYPE pmap[SQL_PDB_MAP_SIZE];
  unsigned short  sqld;
  struct sqlpartkey sqlpartkey[SQL_MAX_NUM_PART_KEYS];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLPARTKEY */
/* ... */
SQL_STRUCTURE sqlpartkey
{
  unsigned short  sqltype;
  unsigned short  sqllen;
};
/* ... */
```

SQLXA-XID

Used by the transaction APIs to identify XA transactions.

Table 70. Fields in the SQLXA-XID Structure

Field Name	Data Type	Description
FORMATID	INTEGER	XA format ID.
GTRID_LENGTH	INTEGER	Length of the global transaction ID.
BQUAL_LENGTH	INTEGER	Length of the branch identifier.

Table 70. Fields in the SQLXA-XID Structure (continued)

Field Name	Data Type	Description
DATA	CHAR[128]	GTRID, followed by BQUAL and trailing blanks, for a total of 128 bytes.
Note: The maximum size for GTRID and BQUAL is 64 bytes each.		

Language syntax:

C Structure

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID */
/* ... */
typedef struct sqlxa_xid_t SQLXA_XID;
/* ... */
```

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID-T */
/* ... */
struct sqlxa_xid_t
{
  sqlint32 formatID;
  sqlint32 gtrid_length;
  sqlint32 bqual_length;
  char data[SQLXA_XIDDATASIZE];
};
/* ... */
```

Appendix A. Naming Conventions

This section provides information about the conventions that apply when naming database manager objects, such as databases and tables, and authentication IDs.

- Character strings that represent names of database manager objects can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.
- Unless otherwise noted, names can be entered in lowercase letters; however, the database manager processes them as if they were uppercase.

  The exception to this is character strings that represent names under the systems network architecture (SNA). Many values, such as logical unit names (partner_lu and local_lu), are case sensitive. The name must be entered exactly as it appears in the SNA definitions that correspond to those terms.
- A database name or database alias is a unique character string containing from one to eight letters, numbers, or keyboard characters from the set described above.

  Databases are cataloged in the system and local database directories by their aliases in one field, and their original name in another. For most functions, the database manager uses the name entered in the alias field of the database directories. (The exceptions are CHANGE DATABASE COMMENT and CREATE DATABASE, where a directory path must be specified.)
- The name or the alias name of a table or a view is an SQL identifier that is a unique character string 1 to 128 characters in length. Column names can be 1 to 30 characters in length.

  A fully qualified table name consists of the *schema.tablename*. The schema is the unique user ID under which the table was created. The schema name for a declared temporary table must be SESSION.
- Authentication IDs cannot exceed 30 characters on Windows 32-bit operating systems and 8 characters on all other operating systems.
- Group IDs cannot exceed 8 characters in length.
- Local aliases for remote nodes that are to be cataloged in the node directory cannot exceed eight characters in length.

Appendix B. Heuristic APIs

Heuristic APIs

Databases can be used in a distributed transaction processing (DTP) environment.

A set of APIs is provided for tool writers to perform heuristic functions on indoubt transactions when the resource owner (such as the database administrator) cannot wait for the Transaction Manager (TM) to perform the *re-sync* action. This condition may occur if, for example, the communication line is broken, and an indoubt transaction is tying up needed resources. For the database manager, these resources include locks on tables and indexes, log space, and storage used by the transaction. Each indoubt transaction also decreases, by one, the maximum number of concurrent transactions that could be processed by the database manager.

The heuristic APIs have the capability to query, commit, and roll back indoubt transactions, and to cancel transactions that have been heuristically committed or rolled back, by removing the log records and releasing log pages.

Attention: The heuristic APIs should be used with caution and only as a last resort. The TM should drive the re-sync events. If the TM has an operator command to start the re-sync action, it should be used. If the user cannot wait for a TM-initiated re-sync, heuristic actions are necessary.

Although there is no set way to perform these actions, the following guidelines may be helpful:

- Use the db2XaListIndTrans function to display the indoubt transactions. They have a status = ’P’ (prepared), and are not connected. The *gtrid* portion of an *xid* is the global transaction ID that is identical to that in other resource managers (RM) that participate in the global transaction.
- Use knowledge of the application and the operating environment to identify the other participating RMs.
- If the transaction manager is CICS, and the only RM is a CICS® resource, perform a heuristic rollback.
- If the transaction manager is not CICS, use it to determine the status of the transaction that has the same *gtrid* as does the indoubt transaction.
- If at least one RM has committed or rolled back, perform a heuristic commit or a rollback.
- If they are all in the prepared state, perform a heuristic rollback.

- If at least one RM is not available, perform a heuristic rollback.

If the transaction manager is available, and the indoubt transaction is due to the RM not being available in the second phase, or in an earlier re-sync, the DBA should determine from the TM's log what action has been taken against the other RMs, and then do the same. The *gtrid* is the matching key between the TM and the RMs.

Do not execute sqlxhfrg unless a heuristically committed or rolled back transaction happens to cause a log full condition. The forget function releases the log space occupied by this indoubt transaction. If a transaction manager eventually performs a re-sync action for this indoubt transaction, the TM could make the wrong decision to commit or to roll back other RMs, because no record was found in this RM. In general, a missing record implies that the RM has rolled back.

db2XaGetInfo - Get Information for Resource Manager

Extracts information for a particular resource manager once an xa_open call has been made.

Authorization:

None

Required Connection:

Database

API Include File:

sqlxa.h

C API Syntax:

```
/* File: sqlxa.h */
/* API: Get Information for Resource Manager */
/* ... */
SQL_API_RC SQL_API_FN
db2XaGetInfo (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2XaGetInfoStruct
{
  db2int32 iRmid;
  struct sqlca oLastSqlca;
} db2XaGetInfoStruct;
/* ... */
```

API Parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2XaGetInfoStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

iRmid Input. Specifies the resource manager for which information is required.

oLastSqlca
Output. Contains the *sqlca* for the last XA API call.

Note: Only the *sqlca* that resulted from the last failing XA API can be retrieved.

Related reference:

- "SQLCA" on page 478

db2XaListIndTrans - List Indoubt Transactions

Provides a list of all indoubt transactions for the currently connected database.

Scope:

This API only affects the database partition on which it is issued.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

db2ApiDf.h

C API syntax:

```
/* File: db2ApiDf.h */
/* API: List Indoubt Transactions */
/* ... */
SQL_API_RC SQL_API_FN
db2XaListIndTrans (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2XaListIndTransStruct
{
db2XaRecoverStruct * piIndoubtData;
db2Uint32            iIndoubtDataLen;
db2Uint32            oNumIndoubtsReturned;
db2Uint32            oNumIndoubtsTotal;
db2Uint32            oReqBufferLen;
} db2XaListIndTransStruct;

typedef SQL_STRUCTURE db2XaRecoverStruct
{
sqluint32      timestamp;
SQLXA_XID   xid;
char           dbalias[SQLXA_DBNAME_SZ];
char           applid[SQLXA_APPLID_SZ];
char           sequence_no[SQLXA_SEQ_SZ];
char           auth_id[SQL_USERID_SZ];
char           log_full;
char           indoubt_status;
char           originator;
char           reserved[8];
} db2XaRecoverStruct;
```

API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2XaListIndTransStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure.

piIndoubtData
Input. A pointer to the application supplied buffer where indoubt data will be returned. The indoubt data is in *db2XaRecoverStruct* format. The application can traverse the list of indoubt transactions by using the size of the *db2XaRecoverStruct* structure, starting at the address provided by this parameter.

If the value is NULL, DB2 will calculate the size of the buffer required and return this value in *oReqBufferLen*. *oNumIndoubtsTotal* will contain the total number of indoubt transactions. The application may allocate the required buffer size and issue the API again.

oNumIndoubtsReturned
Output. The number of indoubt transaction records returned in the buffer specified by *pIndoubtData*.

oNumIndoubtsTotal
Output. The Total number of indoubt transaction records available at the time of API invocation. If the *piIndoubtData* buffer is too small to contain all the records, *oNumIndoubtsTotal* will be greater than the total for *oNumIndoubtsReturned*. The application may reissue the API in order to obtain all records.

Note: This number may change between API invocations as a result of automatic or heuristic indoubt transaction resynchronization, or as a result of other transactions entering the indoubt state.

oReqBufferLen
Output. Required buffer length to hold all indoubt transaction records at the time of API invocation. The application can use this value to determine the required buffer size by calling the API with *pIndoubtData* set to NULL. This value can then be used to allocate the required buffer, and the API can be issued with *pIndoubtData* set to the address of the allocated buffer.

Note: The required buffer size may change between API invocations as a result of automatic or heuristic indoubt transaction resynchronization, or as a result of other transactions entering the indoubt state. The application may allocate a larger buffer to account for this.

timestamp
Output. Specifies the time when the transaction entered the indoubt state.

xid Output. Specifies the XA identifier assigned by the transaction manager to uniquely identify a global transaction.

dbalias
Output. Specifies the alias of the database where the indoubt transaction is found.

applid Output. Specifies the application identifier assigned by the database manager for this transaction.

sequence_no
Output. Specifies the sequence number assigned by the database manager as an extension to the *applid*.

auth_id
Output. Specifies the authorization ID of the user who ran the transaction.

log_full
Output. Indicates whether or not this transaction caused a log full condition. Valid values are:

SQLXA_TRUE
This indoubt transaction caused a log full condition.

SQLXA_FALSE
This indoubt transaction did not cause a log full condition.

indoubt_status
Output. Indicates the status of this indoubt transaction. Valid values are:

SQLXA_TS_PREP
The transaction is prepared. The connected parameter can be used to determine whether the transaction is waiting for the second phase of normal commit processing or whether an error occurred and resynchronization with the transaction manager is required.

SQLXA_TS_HCOM
The transaction has been heuristically committed.

SQLXA_TS_HROL
The transaction has been heuristically rolled back.

SQLXA_TS_MACK
The transaction is missing commit acknowledgement from a node in a partitioned database.

SQLXA_TS_END
The transaction has ended at this database. This transaction may be re-activated, committed, or rolled back at a later time. It is also possible that the transaction manager encountered an error and the transaction will not be completed. If this is the case, this transaction requires heuristic actions, because it may be holding locks and preventing other applications from accessing data.

Usage notes:

A typical application will perform the following steps after setting the current connection to the database or to the partitioned database coordinator node:

1. Call **db2XaListIndTrans** with *piIndoubtData* set to NULL. This will return values in *oReqBufferLen* and *oNumIndoubtsTotal*.
2. Use the returned value in *oReqBufferLen* to allocate a buffer. This buffer may not be large enough if there are additional indoubt transactions because the initial invocation of this API to obtain *oReqBufferLen*. The application may provide a buffer larger than *oReqBufferLen*.
3. Determine if all indoubt transaction records have been obtained. This can be done by comparing *oNumIndoubtsReturned* to *oNumIndoubtTotal*. If *oNumIndoubtsTotal* is greater than *oNumIndoubtsReturned*, the application can repeat the above steps.

Related reference:

- "SQLCA" on page 478
- "sqlxphcm - Commit an Indoubt Transaction" on page 544
- "sqlxphrl - Roll Back an Indoubt Transaction" on page 545

sqlxhfrg - Forget Transaction Status

Permits the RM to erase knowledge of a heuristically completed transaction (that is, one that has been committed or rolled back heuristically).

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqlxa.h

C API syntax:

```
/* File: sqlxa.h */
/* API: Forget Transaction Status */
/* ... */
extern int SQL_API_FN sqlxhfrg(
```

sqlxhfrg - Forget Transaction Status

```
   SQLXA_XID           *pTransId,
   struct sqlca        *pSqlca
   );
/* ... */
```

API parameters:

pTransId
Input. XA identifier of the transaction to be heuristically forgotten, or removed from the database log.

pSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

Only transactions with a status of *heuristically committed* or *rolled back* can have the FORGET operation applied to them.

Related reference:

- "SQLCA" on page 478
- "SQLXA-XID" on page 533

sqlxphcm - Commit an Indoubt Transaction

Commits an indoubt transaction (that is, a transaction that is prepared to be committed). If the operation succeeds, the transaction's state becomes *heuristically committed*.

Scope:

This API only affects the node on which it is issued.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqlxa.h

C API syntax:

```
/* File: sqlxa.h */
/* API: Commit an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphcm(
   int                 exe_type,
   SQLXA_XID           *pTransId,
   struct sqlca        *pSqlca
   );
/* ... */
```

API parameters:

exe_type
: Input. If `EXE_THIS_NODE` is specified, the operation is executed only at this node.

pTransId
: Input. XA identifier of the transaction to be heuristically committed.

pSqlca
: Output. A pointer to the *sqlca* structure.

Usage notes:

Only transactions with a status of *prepared* can be committed. Once heuristically committed, the database manager remembers the state of the transaction until sqlxhfrg is issued.

Related reference:

- "SQLCA" on page 478
- "SQLXA-XID" on page 533
- "sqlxhfrg - Forget Transaction Status" on page 543

sqlxphrl - Roll Back an Indoubt Transaction

Rolls back an indoubt transaction (that is, a transaction that has been prepared). If the operation succeeds, the transaction's state becomes *heuristically rolled back*.

Scope:

This API only affects the node on which it is issued.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqlxa.h

C API syntax:

```
/* File: sqlxa.h */
/* API: Roll Back an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphrl(
   int                  exe_type,
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API parameters:

exe_type
: Input. If `EXE_THIS_NODE` is specified, the operation is executed only at this node.

pTransId
: Input. XA identifier of the transaction to be heuristically rolled back.

pSqlca
: Output. A pointer to the *sqlca* structure.

Usage notes:

Only transactions with a status of *prepared* or *idle* can be rolled back. Once heuristically rolled back, the database manager remembers the state of the transaction until sqlxhfrg is issued.

Related reference:
- "SQLCA" on page 478
- "SQLXA-XID" on page 533
- "sqlxhfrg - Forget Transaction Status" on page 543

Appendix C. Precompiler Customization APIs

There is a set of precompiler service APIs which enable the customization of precompilers. Information about what these APIs are, and how to use them, is available from an anonymous FTP site called **ftp://ftp.software.ibm.com**. The PostScript file, called `prepapi.psbin`, is located in the directory `/ps/products/db2/info`. This file is in binary format.

If you do not have access to this electronic forum and would like to get a copy of the document, contact IBM Support.

Related reference:

- Appendix J, "Contacting IBM" on page 657

Appendix D. Backup and Restore APIs for Vendor Products

Backup and Restore APIs for Vendor Products

DB2 provides interfaces that can be used by third-party media management products to store and retrieve data for backup and restore operations. This function is designed to augment the backup and restore data targets of diskette, disk, tape, and Tivoli Storage Manager, that are supported as a standard part of DB2.

These third-party media management products will be referred to as vendor products in the remainder of this appendix.

DB2 defines a set of function prototypes that provide a general purpose data interface to backup and restore that can be used by many vendors. These functions are to be provided by the vendor in a shared library on UNIX based systems, or DLL on the Windows operating system. When the functions are invoked by DB2, the shared library or DLL specified by the calling backup or restore routine is loaded and the functions provided by the vendor are called to perform the required tasks.

This appendix is divided into four parts:
- Operational overview of DB2's interaction with vendor products.
- Detailed descriptions of DB2's vendor APIs.
- Details on invoking backup and restore using vendor products.
- Information on the data structures used in the API calls.

Sample files demonstrating the DB2 vendor functionality are located on UNIX platforms in the `sqllib/samples/BARVendor` directory, and on Windows in the `sqllib\samples\BARVendor` directory.

Operational Overview

Five functions are defined to interface DB2 and the vendor product:
- sqluvint - Initialize and Link to Device
- sqluvget - Reading Data from Device
- sqluvput - Writing Data to Device
- sqluvend - Unlink the Device
- sqluvdel - Delete Committed Session

DB2 will call these functions, and they should be provided by the vendor product in a shared library on UNIX based systems, or in a DLL on the Windows operating system.

Note: The shared library or DLL code will be run as part of the database engine code. Therefore, it must be reentrant and thoroughly debugged. An errant function may compromise data integrity of the database.

The sequence of functions that DB2 will call during a specific backup or restore operation depends on:

- The number of sessions that will be utilized.
- Whether it is a backup or a restore operation.
- The PROMPTING mode that is specified on the backup or restore operation.
- The characteristics of the device on which the data is stored.
- The errors that may be encountered during the operation.

Number of Sessions

DB2 supports the backup and restore of database objects using one or more data streams or sessions. A backup or restore using three sessions would require three physical or logical devices to be available. When vendor device support is being used, it is the vendor's functions that are responsible for managing the interface to each physical or logical device. DB2 simply sends or receives data buffers to or from the vendor provided functions.

The number of sessions to be used is specified as a parameter by the application that calls the backup or restore database function. This value is provided in the INIT-INPUT structure used by sqluvint.

DB2 will continue to initialize sessions until the specified number is reached, or it receives an SQLUV_MAX_LINK_GRANT warning return code from an sqluvint call. In order to warn DB2 that it has reached the maximum number of sessions that it can support, the vendor product will require code to track the number of active sessions. Failure to warn DB2 could lead to a DB2 initialize session request that fails, resulting in a termination of all sessions and the failure of the entire backup or restore operation.

When the operation is backup, DB2 writes a media header record at the beginning of each session. The record contains information that DB2 uses to identify the session during a restore operation. DB2 uniquely identifies each session by appending a sequence number to the name of the backup image. The number starts at one for the first session, and is incremented by one each time another session is initiated with an sqluvint call for a backup or a restore operation.

When the backup operation completes successfully, DB2 writes a media trailer to the last session it closes. This trailer includes information that tells DB2 how many sessions were used to perform the backup operation. During a restore operation, this information is used to ensure all the sessions, or data streams, have been restored.

Operation with No Errors, Warnings or Prompting

For backup, the following sequence of calls is issued by DB2 for *each* session.

```
sqluvint, action = SQLUV_WRITE
```

followed by 1 to n

```
sqluvput
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

When DB2 issues an sqluvend call (action SQLUV_COMMIT), it expects the vendor product to appropriately save the output data. A return code of SQLUV_OK to DB2 indicates success.

The DB2-INFO structure, used on the sqluvint call, contains the information required to identify the backup. A sequence number is supplied. The vendor product may choose to save this information. DB2 will use it during restore to identify the backup that will be restored.

For restore, the sequence of calls for each session is:

```
sqluvint, action = SQLUV_READ
```

followed by 1 to n

```
sqluvget
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

The information in the DB2-INFO structure used on the sqluvint call will contain the information required to identify the backup. A sequence number is not supplied. DB2 expects that all backup objects (session outputs committed during a backup) will be returned. The first backup object returned is the object generated with sequence number 1, and all other objects are restored in no specific order. DB2 checks the media tail to ensure that all objects have been processed.

Note: Not all vendor products will keep a record of the names of the backup objects. This is most likely when the backups are being done to tapes, or other media of limited capacity. During the initialization of restore sessions, the identification information can be utilized to stage the necessary backup objects so that they are available when required; this may be most useful when juke boxes or robotic systems are used to store the backups. DB2 will always check the media header (first record in each session's output) to ensure that the correct data is being restored.

PROMPTING Mode

When a backup or a restore operation is initiated, two prompting modes are possible:

- WITHOUT PROMPTING or NOINTERRUPT, where there is no opportunity for the vendor product to write messages to the user, or for the user to respond to them.
- PROMPTING or INTERRUPT, where the user can receive and respond to messages from the vendor product.

For PROMPTING mode, backup and restore define three possible user responses:

- Continue

  The operation of reading or writing data to the device will resume.
- Device terminate

  The device will receive no additional data, and the session is terminated.
- Terminate

  The entire backup or restore operation is terminated.

The use of the PROMPTING and WITHOUT PROMPTING modes is discussed in the sections that follow.

Device Characteristics

For purposes of the vendor device support APIs, two general types of devices are defined:

- Limited capacity devices requiring user action to change the media; for example, a tape drive, diskette, or CDROM drive.
- Very large capacity devices, where normal operations do not require the user to handle media; for example, a juke box, or an intelligent robotic media handling device.

A limited capacity device may require that the user be prompted to load additional media during the backup or restore operation. Generally DB2 is not sensitive to the order in which the media is loaded for either backup or restore operations. It also provides facilities to pass vendor media handling messages to the user. This prompting requires that the backup or restore operation be initiated with PROMPTING on. The media handling message text is specified in the description field of the return code structure.

If PROMPTING is on, and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a sqluvput (write) or a sqluvget (read) call, DB2:

- Marks the last buffer sent to the session to be resent, if the call was sqluvput. It will be put to a session later.
- Calls the session with sqluvend (action = SQLUV_COMMIT). If successful (SQLUV_OK return code), DB2:
  - Sends a vendor media handling message to the user from the return code structure that signaled the end-of-media condition.
  - Prompts the user for a continue, device terminate, or terminate response.
- If the response is *continue*, DB2 initializes another session using the sqluvint call, and if successful, begins writing data to or reading data from the session. To uniquely identify the session when writing, DB2 increments the sequence number. The sequence number is available in the DB2-INFO structure used with sqluvint, and is in the media header record, which is the first data record sent to the session.

  DB2 will not start more sessions than requested when a backup or a restore operation is started, or indicated by the vendor product with a SQLUV_MAX_LINK_GRANT warning on an sqluvint call.
- If the response is *device terminate*, DB2 does not attempt to initialize another session, and the number of active sessions is reduced by one. DB2 does not allow all sessions to be terminated by device terminate responses; at least one session must be kept active until the backup or the restore operation completes.
- If the response is *terminate*, DB2 terminates the backup or the restore operation. For more information on exactly what DB2 does to terminate the sessions, see “If Error Conditions Are Returned to DB2” on page 554.

Because backup or restore performance is often dependent on the number of devices being used, it is important that parallelism be maintained. For backup operations, users are encouraged to respond with a `continue`, unless they know that the remaining active sessions will hold the data that is still to be written out. For restore operations, users are also encouraged to respond with a `continue` until all media have been processed.

If the backup or the restore mode is WITHOUT PROMPTING, and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a session, it will terminate the session and not attempt to open another session. If all sessions return end-of-media to DB2 before the backup or the restore operation is complete, the operation will fail. Because of this, WITHOUT PROMPTING should be used carefully with limited capacity devices; it does, however, make sense to operate in this mode with very large capacity devices.

It is possible for the vendor product to hide media mounting and switching actions from DB2, so that the device appears to have infinite capacity. Some very large capacity devices operate in this mode. In these cases, it is critical that all the data that was backed up be returned to DB2 in the same order when a restore operation is in progress. Failure to do so could result in missing data, but DB2 assumes a successful restore operation, because it has no way of detecting the missing data.

DB2 writes data to the vendor product with the assumption that each buffer will be contained on one and only one media (for example, a tape). It is possible for the vendor product to split these buffers across multiple media without DB2's knowledge. In this case, the order in which the media is processed during a restore operation is critical, because the vendor product will be responsible for returning reconstructed buffers from the multiple media to DB2. Failure to do so will result in a failed restore operation.

If Error Conditions Are Returned to DB2

When performing a backup or a restore operation, DB2 expects that all sessions will complete successfully; otherwise, the entire backup or restore operation fails. A session signals successful completion to DB2 with an SQLUV_OK return code on the sqluvend call, action = SQLUV_COMMIT.

If unrecoverable errors are encountered, the session is terminated by DB2. These can be DB2 errors, or errors returned to DB2 from the vendor product. Because all sessions must commit successfully to have a complete backup or restore operation, the failure of one causes DB2 to terminate the other sessions associated with the operation.

If the vendor product responds to a call from DB2 with an unrecoverable return code, the vendor product can optionally provide additional information, using message text placed in the description field of the RETURN-CODE structure. This message text is presented to the user, along with the DB2 information, so that corrective action can be taken.

There will be backup scenarios in which a session has committed successfully, and another session associated with the backup operation experiences an unrecoverable error. Because all sessions must complete successfully before a backup operation is considered successful, DB2 must delete the output data in the committed sessions: DB2 issues a sqluvdel call to request deletion of the object. This call is not considered an I/O session, and is responsible for initializing and terminating any connection that may be necessary to delete the backup object.

The DB2-INFO structure will not contain a sequence number; sqluvdel will delete all backup objects that match the remaining parameters in the DB2-INFO structure.

Warning Conditions

It is possible for DB2 to receive warning return codes from the vendor product; for example, if a device is not ready, or some other correctable condition has occurred. This is true for both read and write operations.

On sqluvput and sqluvget calls, the vendor can set the return code to SQLUV_WARNING, and optionally provide additional information, using message text placed in the description field of the RETURN-CODE structure. This message text is presented to the user so that corrective action can be taken. The user can respond in one of three ways: continue, device terminate, or terminate:

- If the response is *continue*, DB2 attempts to rewrite the buffer using sqluvput during a backup operation. During a restore operation, DB2 issues an sqluvget call to read the next buffer.
- If the response is *device terminate* or *terminate*, DB2 terminates the entire backup or restore operation in the same way that it would respond after an unrecoverable error (for example, it will terminate active sessions and delete committed sessions).

Operational Hints and Tips

This section provides some hints and tips for building vendor products.

History File

The history file can be used as an aid in database recovery operations. It is associated with each database, and is automatically updated with each backup or restore operation. Information in the file can be viewed, updated, or pruned through the following facilities:

- Control Center
- Command line processor (CLP)
  - LIST HISTORY command
  - UPDATE HISTORY FILE command
  - PRUNE HISTORY command
- APIs
  - db2HistoryOpenScan
  - db2HistoryGetEntry
  - db2HistoryCloseScan
  - db2HistoryUpdate
  - db2Prune

For information about the layout of the file, see db2HistData.

When a backup operation completes, one or more records is written to the file. If the output of the backup operation was directed to vendor devices and the LOAD keyword was used, the DEVICE field in the history record contains an O. If the backup operation was directed to TSM, the DEVICE field contains an A. The LOCATION field contains either:

- The vendor file name specified when the backup operation was invoked.
- The name of the shared library, if no vendor file name was specified.

For more information about specifying this option, see "Invoking a Backup or a Restore Operation Using Vendor Products".

The LOCATION field can be updated using the Control Center, the CLP, or an API. The location of backup information can be updated if limited capacity devices (for example, removable media) have been used to hold the backup image, and the media is physically moved to a different (perhaps off-site) storage location. If this is the case, the history file can be used to help locate a backup image if a recovery operation becomes necessary.

Invoking a Backup or a Restore Operation Using Vendor Products

Vendor products can be specified when invoking the DB2 backup or the DB2 restore utility from:

- The Control Center
- The command line processor (CLP)
- An application programming interface (API).

The Control Center

The Control Center is the graphical user interface for database administration that is shipped with DB2.

To specify	The Control Center input variable for backup or restore operations
Use of vendor device and library name	Is *Use Library*. Specify the library name (on UNIX based systems) or the DLL name (on the Windows operating system).
Number of sessions	Is *Sessions*.
Vendor options	Is not supported.
Vendor file name	Is not supported.
Transfer buffer size	Is (for backup) *Size of each Buffer*, and (for restore) not applicable.

The Command Line Processor (CLP)

The command line processor (CLP) can be used to invoke the DB2 BACKUP DATABASE or the RESTORE DATABASE command.

To specify	**The command line processor parameter**	
	for backup is	**for restore is**
Use of vendor device and library name	*library-name*	*shared-library*
Number of sessions	*num-sessions*	*num-sessions*
Vendor options	not supported	not supported
Vendor file name	not supported	not supported
Transfer buffer size	*buffer-size*	*buffer-size*

Backup and Restore API Function Calls

Two API function calls support backup and restore operations: db2Backup for backup and db2Restore for restore.

To specify	**The API parameter (for both db2Backup and db2Restore) is**
Use of vendor device and library name	as follows: In structure *sqlu_media_list*, specify a media type of SQLU_OTHER_MEDIA, and then in structure *sqlu_vendor*, specify a shared library or DLL in *shr_lib*.
Number of sessions	as follows: In structure *sqlu_media_list*, specify *sessions*.
Vendor options	*PVendorOptions*
Vendor file name	as follows: In structure *sqlu_media_list*, specify a media type of SQLU_OTHER_MEDIA, and then in structure *sqlu_vendor*, specify a file name in *filename*.
Transfer buffer size	*BufferSize*

Related reference:

- "sqluvint - Initialize and Link to Device" on page 558
- "sqluvget - Reading Data from Device" on page 561
- "sqluvput - Writing Data to Device" on page 563
- "sqluvend - Unlink the Device and Release its Resources" on page 565
- "sqluvdel - Delete Committed Session" on page 568
- "DB2-INFO" on page 569
- "VENDOR-INFO" on page 571
- "INIT-INPUT" on page 573

- "INIT-OUTPUT" on page 574
- "DATA" on page 574
- "RETURN-CODE" on page 575

sqluvint - Initialize and Link to Device

This function is called to provide information for initialization and establishment of a logical link between DB2 and the vendor device.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API parameters:

Init_input
Input. Structure that contains information provided by DB2 to establish a logical link with the vendor device.

Init_output
Output. Structure that contains the output returned by the vendor device.

Return_code
Output. Structure that contains the return code to be passed to DB2, and a brief text explanation.

Usage notes:

For each media I/O session, DB2 will call this function to obtain a device handle. If for any reason, the vendor function encounters an error during initialization, it will indicate it via a return code. If the return code indicates an error, DB2 may choose to terminate the operation by calling the **sqluvend** function. Details on possible return codes, and the DB2 reaction to each of these, is contained in the return codes table (see Table 71 on page 560).

The INIT-INPUT structure contains elements that can be used by the vendor product to determine if the backup or restore can proceed:

- size_HI_order and size_LOW_order

  This is the estimated size of the backup. They can be used to determine if the vendor devices can handle the size of the backup image. They can be used to estimate the quantity of removable media that will be required to hold the backup. It might be beneficial to fail at the first **sqluvint** call if problems are anticipated.
- req_sessions

  The number of user requested sessions can be used in conjunction with the estimated size and the prompting level to determine if the backup or restore operation is possible.
- prompt_lvl

  The prompting level indicates to the vendor if it is possible to prompt for actions such as changing removable media (for example, put another tape in the tape drive). This might suggest that the operation cannot proceed since there will be no way to prompt the user.

  If the prompting level is WITHOUT PROMPTING and the quantity of removable media is greater than the number of sessions requested, DB2 will not be able to complete the operation successfully.

DB2 names the backup being written or the restore to be read via fields in the DB2-INFO structure. In the case of an action = SQLUV_READ, the vendor product must check for the existence of the named object. If it cannot be found, the return code should be set to SQLUV_OBJ_NOT_FOUND so that DB2 will take the appropriate action.

After initialization is completed successfully, DB2 will continue by issuing other data transfer functions, but may terminate the session at any time with an **sqluvend** call.

Return codes:

Table 71. Valid Return Codes for sqluvint and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput, sqluvget (see comments)	If action = SQLUV_WRITE, the next call will be sqluvput (to BACKUP data). If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_OK; the next call will be sqluvget to RESTORE data.
SQLUV_LINK_EXIST	Session activated previously.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_COMM_ ERROR	Communication error with device.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_VERSION	The DB2 and vendor products are incompatible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ACTION	Invalid action is requested. This could also be used to indicate that the combination of parameters results in an operation which is not possible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NO_DEV_ AVAIL	No device is available for use at the moment.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJ_NOT_ FOUND	Object specified cannot be found. This should be used when the action on the sqluvint call is 'R' (read) and the requested object cannot be found based on the criteria specified in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJS_FOUND	More than 1 object matches the specified criteria. This will result when the action on the sqluvint call is 'R' (read) and more than one object matches the criteria in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_USERID	Invalid userid specified.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ PASSWORD	Invalid password provided.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

Table 71. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_INV_OPTIONS	Invalid options encountered in the vendor options field.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INIT_FAILED	Initialization failed and the session is to be terminated.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_DEV_ERROR	Device error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_MAX_LINK_ GRANT	Max number of links established.	sqluvput, sqluvget (see comments)	This is treated as a warning by DB2. The warning tells DB2 not to open additional sessions with the vendor product, because the maximum number of sessions it can support has been reached (note: this could be due to device availability). If action = SQLUV_WRITE (BACKUP), the next call will be sqluvput. If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_MAX_LINK_GRANT; the next call will be sqluvget to RESTORE data.
SQLUV_IO_ERROR	I/O error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NOT_ ENOUGH_SPACE	There is not enough space to store the entire backup image; the size estimate is provided as a 64-bit value in bytes.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

sqluvget - Reading Data from Device

After initialization, this function can be called to read data from the device.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqluvend.h

sqluvget - Reading Data from Device

C API syntax:

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data         *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API parameters:

pVendorCB
Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data Input/output. A pointer to the *data* structure.

Return_code
Output. The return code from the API call.

obj_num
Specifies which backup object should be retrieved.

buff_size
Specifies the buffer size to be used.

actual_buff_size
Specifies the actual bytes read or written. This value should be set to output to indicate how many bytes of data were actually read.

dataptr
A pointer to the data buffer.

reserve
Reserved for future use.

Usage notes:

This function is used by the restore utility.

Return codes:

Table 72. Valid Return Codes for sqluvget and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvget	DB2 processes the data
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA or SQLUV_ENDOFMEDIA_NO_ DATA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvget, or sqluvend, action = SQLU_ABORT	
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_MORE_DATA	Operation successful; more data available.	sqluvget	
SQLUV_ENDOFMEDIA_NO_ DATA	End of media and 0 bytes read (for example, end of tape).	sqluvend	
SQLUV_ENDOFMEDIA	End of media and > 0 bytes read, (for example, end of tape).	sqluvend	DB2 processes the data, and then handles the end-of-media condition.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.

Next call:

[a] If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated.

sqluvput - Writing Data to Device

After initialization, this function can be used to write data to the device.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

sqluvput - Writing Data to Device

Database

API include file:

sqluvend.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API parameters:

pVendorCB
: Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data
: Output. Data buffer filled with data to be written out.

Return_code
: Output. The return code from the API call.

obj_num
: Specifies which backup object should be retrieved.

buff_size
: Specifies the buffer size to be used.

actual_buff_size
: Specifies the actual bytes read or written. This value should be set to indicate how many bytes of data were actually read.

dataptr
: A pointer to the data buffer.

reserve
: Reserved for future use.

Usage notes:

This function is used by the backup utility.

Return codes:

Table 73. Valid Return Codes for sqluvput and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput or sqluvend, if complete (for example, DB2 has no more data)	Inform other processes of successful operation.
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_ENDOFMEDIA	End of media reached, for example, end of tape.	sqluvend	
SQLUV_DATA_RESEND	Device requested to have buffer sent again.	sqluvput	DB2 will retransmit the last buffer. This will only be done once.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvput	
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.

Next call:

[a] If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated. Committed sessions are deleted with an sqluvint, sqluvdel, and sqluvend sequence of calls.

sqluvend - Unlink the Device and Release its Resources

Ends or unlinks the device, and frees all of its related resources. The vendor must free or release unused resources (for example, allocated space and file handles) before returning to DB2.

Authorization:

One of the following:
- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sql.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  sqlint32 action,
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API parameters:

action Input. Used to commit or abort the session:
- SQLUV_COMMIT (0 = to commit)
- SQLUV_ABORT (1 = to abort)

pVendorCB
Input. Pointer to the Init_output structure.

Init_output
Output. Space for Init_output de-allocated. The data has been committed to stable storage for a backup if action is to commit. The data is purged for a backup if the action is to abort.

Return code
Output. The return code from the API call.

Usage notes:

This function is called for each session that has been opened. There are two possible action codes:
- Commit

  Output of data to this session, or the reading of data from the session, is complete.

For a write (backup) session, if the vendor returns to DB2 with a return code of SQLUV_OK, DB2 assumes that the output data has been appropriately saved by the vendor product, and can be accessed if referenced in a later **sqluvint** call.

For a read (restore) session, if the vendor returns to DB2 with a return code of SQLUV_OK, the data should not be deleted, because it may be needed again.

If the vendor returns SQLUV_COMMIT_FAILED, DB2 assumes that there are problems with the entire backup or restore operation. All active sessions are terminated by **sqluvend** calls with action = SQLUV_ABORT. For a backup operation, committed sessions receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls.

- Abort

  A problem has been encountered by DB2, and there will be no more reading or writing of data to the session.

  For a write (backup) session, the vendor should delete the partial output dataset, and use a SQLUV_OK return code if the partial output is deleted. DB2 assumes that there are problems with the entire backup. All active sessions are terminated by **sqluvend** calls with action = SQLUV_ABORT, and committed sessions receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls.

  For a read (restore) session, the vendor should not delete the data (because it may be needed again), but should clean up and return to DB2 with a SQLUV_OK return code. DB2 terminates all the restore sessions by **sqluvend** calls with action = SQLUV_ABORT. If the vendor returns SQLUV_ABORT_FAILED to DB2, the caller is not notified of this error, because DB2 returns the first fatal failure and ignores subsequent failures. In this case, for DB2 to have called **sqluvend** with action = SQLUV_ABORT, an initial fatal error must have occurred.

Return codes:

Table 74. Valid Return Codes for sqluvend and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	no further calls	Free all memory allocated for this session and terminate.
SQLUV_COMMIT_FAILED	Commit request failed.	no further calls	Free all memory allocated for this session and terminate.
SQLUV_ABORT_FAILED	Abort request failed.	no further calls	

sqluvdel - Delete Committed Session

Deletes committed sessions.

Authorization:

One of the following:

- *sysadm*
- *dbadm*

Required connection:

Database

API include file:

sqluvend.h

C API syntax:

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API parameters:

Init_input
: Input. Space allocated for Init_input and Return_code.

Return_code
: Output. Return code from the API call. The object pointed to by the Init_input structure is deleted.

Usage notes:

If multiple sessions are opened, and some sessions are committed, but one of them fails, this function is called to delete the committed sessions. No sequence number is specified; **sqluvdel** is responsible for finding all of the objects that were created during a particular backup operation, and deleting them. Information in the INIT-INPUT structure is used to identify the output data to be deleted. The call to **sqluvdel** is responsible for establishing any connection or session that is required to delete a backup object from the vendor device. If the return code from this call is SQLUV_DELETE_FAILED, DB2 does not notify the caller, because DB2 returns the first fatal failure and ignores subsequent failures. In this case, for DB2 to have called **sqluvdel**, an initial fatal error must have occurred.

Return codes:

Table 75. Valid Return Codes for sqluvdel and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	no further calls	
SQLUV_DELETE_FAILED	Delete request failed.	no further calls	

DB2-INFO

This structure contains information identifying DB2 to the vendor device.

Table 76. Fields in the DB2-INFO Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
DB2_id	char	An identifier for the DB2 product. Maximum length of the string it points to is 8 characters.
version	char	The current version of the DB2 product. Maximum length of the string it points to is 8 characters.
release	char	The current release of the DB2 product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
level	char	The current level of the DB2 product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
action	char	Specifies the action to be taken. Maximum length of the string it points to is 1 character.
filename	char	The file name used to identify the backup image. If it is NULL, the *server_id*, *db2instance*, *dbname*, and *timestamp* will uniquely identify the backup image. Maximum length of the string it points to is 255 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of the string it points to is 8 characters.
db2instance	char	The db2instance ID. This is the user ID invoking the command. Maximum length of the string it points to is 8 characters.

Table 76. Fields in the DB2-INFO Structure (continued). All fields are NULL-terminated strings.

Field Name	Data Type	Description

type	char	Specifies the type of backup being taken or the type of restore being performed. The following are possible values:
		
		When action is SQLUV_WRITE:
		`0 - full database backup`
		`3 - table space level backup`
		
		When action is SQLUV_READ:
		`0 - full restore`
		`3 - online table space restore`
		`4 - table space restore`
		`5 - history file restore`

dbname	char	The name of the database to be backed up or restored. Maximum length of the string it points to is 8 characters.
alias	char	The alias of the database to be backed up or restored. Maximum length of the string it points to is 8 characters.
timestamp	char	The time stamp used to identify the backup image. Maximum length of the string it points to is 26 characters.
sequence	char	Specifies the file extension for the backup image. For write operations, the value for the first session is 1 and each time another session is initiated with an sqluvint call, the value is incremented by 1. For read operations, the value is always zero. Maximum length of the string it points to is 3 characters.
obj_list	struct sqlu_gen_list	Reserved for future use.
max_bytes_per_txn	sqlint32	Specifies to the vendor in bytes, the transfer buffer size specified by the user.
image_filename	char	Reserved for future use.
reserve	void	Reserved for future use.
nodename	char	Name of the node at which the backup was generated.
password	char	Password for the node at which the backup was generated.
owner	char	ID of the backup originator.
mcNameP	char	Management class.
nodeNum	SQL_PDB_NODE_TYPE	Node number. Numbers greater than 255 are supported by the vendor interface.

The *filename*, or *server_id*, *db2instance*, *type*, *dbname* and *timestamp* uniquely identifies the backup image. The sequence number, specified by *sequence*, identifies the file extension. When a backup image is to be restored, the same values must be specified to retrieve the backup image. Depending on the vendor product, if *filename* is used, the other parameters may be set to NULL, and vice versa.

Language syntax:

C Structure

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                  *DB2_id;
  char                  *version;
  char                  *release;
  char                  *level;
  char                  *action;
  char                  *filename;
  char                  *server_id;
  char                  *db2instance;
  char                  *type;
  char                  *dbname;
  char                  *alias;
  char                  *timestamp;
  char                  *sequence;
  struct sqlu_gen_list  *obj_list;
  long                  max_bytes_per_txn;
  char                  *image_filename;
  void                  *reserve;
  char                  *nodename;
  char                  *password;
  char                  *owner;
  char                  *mcNameP;
  SQL_PDB_NODE_TYPE     nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

This structure contains information identifying the vendor and version of the device.

Table 77. Fields in the VENDOR-INFO Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
vendor_id	char	An identifier for the vendor. Maximum length of the string it points to is 64 characters.

Table 77. Fields in the VENDOR-INFO Structure (continued). All fields are NULL-terminated strings.

Field Name	Data Type	Description
version	char	The current version of the vendor product. Maximum length of the string it points to is 8 characters.
release	char	The current release of the vendor product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
level	char	The current level of the vendor product. Set to NULL if it is insignificant. Maximum length of the string it points to is 8 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of the string it points to is 8 characters.
max_bytes_per_txn	sqlint32	The maximum supported transfer buffer size. Specified by the vendor, in bytes. This is used only if the return code from the vendor initialize function is SQLUV_BUFF_SIZE, indicating that an invalid buffer size was specified.
num_objects_in_backup	sqlint32	The number of sessions that were used to make a complete backup. This is used to determine when all backup images have been processed during a restore operation.
reserve	void	Reserved for future use.

Language syntax:

C Structure

```
typedef struct Vendor_info
{
  char       *vendor_id;
  char       *version;
  char       *release;
  char       *level;
  char       *server_id;
  sqlint32   max_bytes_per_txn;
  sqlint32   num_objects_in_backup;
  void       *reserve;
} Vendor_info;
```

INIT-INPUT

This structure contains information provided by DB2 to set up and to establish a logical link with the vendor device.

Table 78. Fields in the INIT-INPUT Structure. All fields are NULL-terminated strings.

Field Name	Data Type	Description
DB2_session	struct DB2_info	A description of the session from the perspective of DB2.
size_options	unsigned short	The length of the options field. When using the DB2 backup or restore function, the data in this field is passed directly from the *VendorOptionsSize* parameter.
size_HI_order	sqluint32	High order 32 bits of DB size estimate in bytes; total size is 64 bits.
size_LOW_order	sqluint32	Low order 32 bits of DB size estimate in bytes; total size is 64 bits.
options	void	This information is passed from the application when the backup or the restore function is invoked. This data structure must be flat; in other words, no level of indirection is supported. Byte-reversal is not done, and the code page for this data is not checked. When using the DB2 backup or restore function, the data in this field is passed directly from the *pVendorOptions* parameter.
reserve	void	Reserved for future use.
prompt_lvl	char	Prompting level requested by the user when a backup or a restore operation was invoked. Maximum length of the string it points to is 1 character.
num_sessions	unsigned short	Number of sessions requested by the user when a backup or a restore operation was invoked.

Language syntax:

C Structure

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  sqluint32        size_HI_order;
  sqluint32        size_LOW_order;
  void             *options;
```

INIT-INPUT

```
  void            *reserve;
  char            *prompt_lvl;
  unsigned short   num_sessions;
} Init_input;
```

INIT-OUTPUT

This structure contains the output returned by the vendor device.

Table 79. Fields in the INIT-OUTPUT Structure

Field Name	Data Type	Description
vendor_session	struct Vendor_info	Contains information to identify the vendor to DB2.
pVendorCB	void	Vendor control block.
reserve	void	Reserved for future use.

Language syntax:

C Structure

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

This structure contains data transferred between DB2 and the vendor device.

Table 80. Fields in the DATA Structure

Field Name	Data Type	Description
obj_num	sqlint32	The sequence number assigned by DB2 during a backup operation.
buff_size	sqlint32	The size of the buffer.
actual_buf_size	sqlint32	The actual number of bytes sent or received. This must not exceed *buff_size*.
dataptr	void	Pointer to the data buffer. DB2 allocates space for the buffer.
reserve	void	Reserved for future use.

Language syntax:

C Structure

```
typedef struct Data
{
  sqlint32  obj_num;
  sqlint32  buff_size;
  sqlint32  actual_buff_size;
  void      *dataptr;
  void      *reserve;
} Data;
```

RETURN-CODE

This structure contains the return code and a short explanation of the error being returned to DB2.

Table 81. Fields in the RETURN-CODE Structure

Field Name	Data Type	Description
return_code[a]	sqlint32	Return code from the vendor function.
description	char	A short description of the return code.
reserve	void	Reserved for future use.
[a] This is a vendor-specific return code that is not the same as the value returned by various DB2 APIs. See the individual API descriptions for the return codes that are accepted from vendor products.		

Language syntax:

C Structure

```
typedef struct Return_code
{
  sqlint32  return_code,
  char      description[30],
  void      *reserve,
} Return_code;
```

Appendix E. Threaded Applications with Concurrent Access

Threaded Applications with Concurrent Access

In the default implementation of threaded applications against a DB2 database, serialization of access to the database is enforced by the database APIs. If one thread performs a database call, calls made by other threads will be blocked until the first call completes, even if the subsequent calls access database objects that are unrelated to the first call. In addition, all threads within a process share a commit scope. True concurrent access to a database can only be achieved through separate processes, or by using the APIs that are described in this section.

This section describes APIs that can be used to allocate and manipulate separate environments (contexts) for the use of database APIs and embedded SQL. Each context is a separate entity, and any connection or attachment using one context is independent of all other contexts (and thus all other connections or attachments within a process). In order for work to be done on a context, it must first be associated with a thread. A thread must always have a context when making database API calls or when using embedded SQL. If these APIs to manipulate contexts are not used, all threads within a process share the same context. If these APIs are used, each thread can have its own context. It will have a separate connection to a database or attachment to an instance, and will have its own commit scope.

Contexts need not be associated with a given thread for the duration of a connection or attachment. One thread can attach to a context, connect to a database, detach from the context, and then a second thread can attach to the context and continue doing work using the already existing database connection. Contexts can be passed around among threads in a process, but not among processes.

If the new APIs are not used, the old behavior is in effect, and existing applications need not change.

Even if the new APIs are used, the following APIs continue to be serialized:
- sqlabndx - Bind
- sqlaprep - Precompile Program
- sqluexpr - Export
- sqluimpr - Import.

The new APIs can be used with embedded SQL and the transaction APIs.

These APIs have no effect (that is, they are no-ops) on platforms that do not support application threading.

Notes:

1. The DB2 CLI automatically uses multiple contexts to achieve thread-safe, concurrent database access on platforms that support multi-threading. While not recommended by DB2, users can explicitly disable this feature if required.
2. By default, AIX does not permit 32-bit applications to attach to more than 11 shared memory segments per process, of which a maximum of 10 can be used for DB2 connections.

   When this limit is reached, DB2 returns SQLCODE -1224 on an SQL CONNECT. DB2 Connect also has the 10-connection limitation if local users are running two-phase commit over SNA, or two-phase commit with a TP Monitor (SNA or TCP/IP).

   The AIX environment variable **EXTSHM** can be used to increase the maximum number of shared memory segments to which a process can attach.

   To use EXTSHM with DB2, do the following:

   In client sessions:

   ```
   export EXTSHM=ON
   ```

   When starting the DB2 server:

   ```
   export EXTSHM=ON
   db2set DB2ENVLIST=EXTSHM
   db2start
   ```

   On ESE, also add the following lines to sqllib/db2profile:

   ```
   EXTSHM=ON
   export EXTSHM
   ```

   An alternative is to move the local database or DB2 Connect into another machine and to access it remotely, or to access the local database or the DB2 Connect database with TCP/IP loop-back by cataloging it as a remote node that has the TCP/IP address of the local machine.

Related samples:

- “dbthrds.sqc -- How to use multiple context APIs on UNIX (C)”
- “dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)”

sqleAttachToCtx - Attach to Context

Makes the current thread use a specified context. All subsequent database calls made on this thread will use this context. If more than one thread is attached to a given context, access is serialized for these threads, and they share a commit scope.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
int sqleAttachToCtx (
void          *pCtx,
void          *reserved,
struct sqlca   *pstSqlca);
```

API parameters:

pCtx Input. A valid context previously allocated by sqleBeginCtx.

reserved
Reserved for future use. Must be set to NULL.

pstSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478
- "sqleBeginCtx - Create and Attach to an Application Context" on page 580

Related samples:

- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

sqleBeginCtx - Create and Attach to an Application Context

Creates an application context, or creates and then attaches to an application context. More than one application context can be created. Each context has its own commit scope. Different threads can attach to different contexts (see sqleAttachToCtx). Any database API calls made by such threads will not be serialized with one another.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
int sqleBeginCtx (
void           **ppCtx,
sqlint32       lOptions,
void           *reserved,
struct sqlca   *pstSqlca);
```

API parameters:

ppCtx Output. A data area allocated out of private memory for the storage of context information.

lOptions
Input. Valid values are:

SQL_CTX_CREATE_ONLY
The context memory will be allocated, but there will be no attachment.

SQL_CTX_BEGIN_ALL
The context memory will be allocated, and then a call to sqleAttachToCtx will be made for the current thread. If this option is used, the *ppCtx* parameter can be NULL. If the thread is already attached to a context, the call will fail.

reserved
Reserved for future use. Must be set to NULL.

pstSqlca
Output. A pointer to the *sqlca* structure.

Related reference:
- "SQLCA" on page 478
- "sqleAttachToCtx - Attach to Context" on page 579

Related samples:
- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

sqleDetachFromCtx - Detach From Context

Detaches the context being used by the current thread. The context will be detached only if an attach to that context has previously been made.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
int sqleDetachFromCtx (
void          *pCtx,
void          *reserved,
struct sqlca  *pstSqlca);
```

API parameters:

pCtx Input. A valid context previously allocated by sqleBeginCtx.

reserved
Reserved for future use. Must be set to NULL.

pstSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478
- "sqleBeginCtx - Create and Attach to an Application Context" on page 580

Related samples:

- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

sqleEndCtx - Detach and Destroy Application Context

Frees all memory associated with a given context.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
int sqleEndCtx (
void          **ppCtx,
sqlint32      lOptions,
void          *reserved,
struct sqlca  *pstSqlca);
```

API parameters:

ppCtx Output. A data area in private memory (used for the storage of context information) that is freed.

lOptions
Input. Valid values are:

SQL_CTX_FREE_ONLY
The context memory will be freed only if a prior detach has been done.

Note: *pCtx* must be a valid context previously allocated by sqleBeginCtx.

SQL_CTX_END_ALL
If necessary, a call to sqleDetachFromCtx will be made before the memory is freed.

Note: A detach will be done even if the context is still in use. If this option is used, the *ppCtx* parameter can be NULL, but if passed, it must be a valid context previously allocated by sqleBeginCtx. A call to sqleGetCurrentCtx will be made, and the current context freed from there.

reserved
Reserved for future use. Must be set to NULL.

pstSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

If a database connection exists, or the context has been attached by another thread, this call will fail.

Note: If a context calls an API that establishes an instance attachment (for example, db2CfgGet, it is necessary to detach from the instance using sqledtin before calling sqleEndCtx.

Related reference:
- "sqledtin - Detach" on page 345
- "SQLCA" on page 478
- "sqleBeginCtx - Create and Attach to an Application Context" on page 580
- "sqleDetachFromCtx - Detach From Context" on page 581
- "sqleGetCurrentCtx - Get Current Context" on page 584
- "db2CfgGet - Get Configuration Parameters" on page 39

sqleGetCurrentCtx - Get Current Context

Returns the current context associated with a thread.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
int sqleGetCurrentCtx (
void           **ppCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API parameters:

ppCtx Output. A data area allocated out of private memory for the storage of context information.

h reserved
Reserved for future use. Must be set to NULL.

pstSqlca
Output. A pointer to the *sqlca* structure.

Related reference:

- "SQLCA" on page 478

sqleInterruptCtx - Interrupt Context

Interrupts the specified context.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

Database

API include file:

sql.h

C API syntax:

```
int sqleInterruptCtx (
void          *pCtx,
void          *reserved,
struct sqlca  *pstSqlca);
```

API parameters:

pCtx Input. A valid context previously allocated by sqleBeginCtx.

reserved
Reserved for future use. Must be set to NULL.

pstSqlca
Output. A pointer to the *sqlca* structure.

Usage notes:

During processing, this API:
- Switches to the context that has been passed in
- Sends an interrupt
- Switches to the original context
- Exits.

Related reference:
- "SQLCA" on page 478

- "sqleBeginCtx - Create and Attach to an Application Context" on page 580

sqleSetTypeCtx - Set Application Context Type

Sets the application context type. This API should be the first database API called inside an application.

Scope:

The scope of this API is limited to the immediate process.

Authorization:

None

Required connection:

None

API include file:

sql.h

C API syntax:

```
int sqleSetTypeCtx (
sqlint32   lOptions);
```

API parameters:

lOptions
Input. Valid values are:

SQL_CTX_ORIGINAL
All threads will use the same context, and concurrent access will be blocked. This is the default if none of these APIs is called.

SQL_CTX_MULTI_MANUAL
All threads will use separate contexts, and it is up to the application to manage the context for each thread. See

- sqleBeginCtx
- sqleAttachToCtx
- sqleDetachFromCtx
- sqleEndCtx

The following restrictions/changes apply when this option is used:

- When termination is normal, automatic COMMIT at process termination is disabled. All outstanding transactions are rolled back, and all COMMITs must be done explicitly.
- sqleintr interrupts all contexts. To interrupt a specific context, use sqleInterruptCtx.

Usage notes:

This API must be called *before* any other database call, and only the first call is effective.

Related reference:
- "sqleintr - Interrupt" on page 364
- "sqleAttachToCtx - Attach to Context" on page 579
- "sqleBeginCtx - Create and Attach to an Application Context" on page 580
- "sqleDetachFromCtx - Detach From Context" on page 581
- "sqleEndCtx - Detach and Destroy Application Context" on page 582
- "sqleInterruptCtx - Interrupt Context" on page 585

Related samples:
- "dbthrds.sqc -- How to use multiple context APIs on UNIX (C)"
- "dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)"

Appendix F. DB2 UDB Log Records

This section describes the structure of the DB2 UDB log records returned by the db2ReadLog API.

All DB2 UDB log records begin with a log manager header. This header includes the total log record size, the log record type, and transaction-specific information. It does not include information about accounting, statistics, traces, or performance evaluation. For more information, see "Log Manager Header" on page 591.

Log records are uniquely identified by a log sequence number (LSN). The LSN represents a relative byte address, within the database log, for the first byte of the log record. It marks the offset of the log record from the beginning of the database log.

The log records written by a single transaction are uniquely identifiable by a field in the log record header. The unique transaction identifier is a six-byte field that increments by one whenever a new transaction is started. All log records written by a single transaction contain the same identifier.

When a transaction performs writable work against a table with DATA CAPTURE CHANGES on, or invokes a log writing utility, the transaction is marked as propagatable. Only propagatable transactions have their transaction manager log records marked as propagatable.

Table 82. DB2 UDB Log Records

Data Manager	
"Initialize Table" on page 596	New permanent table creation.
"Import Replace (Truncate)" on page 598	Import replace activity.
"Rollback Insert" on page 598	Rollback row insert.
"Reorg Table" on page 599	REORG committed.
"Create Index, Drop Index" on page 599	Index activity.
"Create Table, Drop Table, Rollback Create Table, Rollback Drop Table" on page 600	Table activity.
"Alter Table Attribute" on page 600	Propagation, check pending, and append mode activity.
"Alter Table Add Columns, Rollback Add Columns" on page 601	Adding columns to existing tables.

Table 82. DB2 UDB Log Records (continued)

"Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record" on page 602	Table record activity.
"Update Record" on page 607	Row updates where storage location not changed.
Long Field Manager	
"Add/Delete/Non-update Long Field Record" on page 609	Long field record activity.
Transaction Manager	
"Normal Commit" on page 609	Transaction commits.
"Heuristic Commit" on page 610	Indoubt transaction commits.
"MPP Coordinator Commit" on page 610	Transaction commits. This is written on a coordinator node for an application that performs updates on at least one subordinator node.
"MPP Subordinator Commit" on page 611	Transaction commits. This is written on a subordinator node.
"Normal Abort" on page 611	Transaction aborts.
"Heuristic Abort" on page 611	Indoubt transaction aborts.
"Local Pending List" on page 612	Transaction commits with a pending list existing.
"Global Pending List" on page 612	Transaction commits (two-phase) with a pending list existing.
"XA Prepare" on page 613	XA transaction preparation in two-phase commit environments.
"MPP Subordinator Prepare" on page 613	MPP transaction preparation in two-phase commit environments. This log record only exists on subordinator nodes.
"Backout Free" on page 614	Marks the end of a backout free interval. The backout free interval is a set of log records that is not to be compensated if the transaction aborts.
Utility Manager	
"Migration Begin" on page 615	Catalog migration starts.
"Migration End" on page 615	Catalog migration completes.
"Load Start" on page 615	Table load starts.
"Table Load Delete Start" on page 615	Load delete phase starts.
"Load Delete Start Compensation" on page 616	Load delete phase ends.

Table 82. DB2 UDB Log Records (continued)

"Load Pending List" on page 616	Table load completes.
"Backup End" on page 616	Backup activity completes.
"Table Space Rolled Forward" on page 616	Table space rollforward completes.
"Table Space Roll Forward to PIT Begins" on page 617	Marks the beginning of a table space rollforward to a point in time.
"Table Space Roll Forward to PIT Ends" on page 617	Marks the end of a table space rollforward to a point in time.
Datalink Manager	
"Link File" on page 618	Written when an insert or an update on a table with a DATALINK column creates a link to a file.
"Unlink File" on page 619	Written when a delete or an update on a table with a DATALINK column drops a link to a file.
"Delete Group" on page 620	Written when a table with DATALINK columns (having the file link control attribute) is dropped.
"Delete PGroup" on page 621	Written when a table space is dropped.
"DLFM Prepare" on page 621	Written during the prepare phase, when a two-phase commit is used for transactions involving DB2 Data Links Manager.

Log Manager Header

All DB2 UDB log records begin with a log manager header. This header contains information detailing the log record and transaction information of the log record writer.

Note: A log record of type 'i' is an informational log record only. It will be ignored by DB2 during rollforward, rollback, and crash recovery.

Table 83. Log Manager Log Record Header (LogManagerLogRecordHeader)

Description	Type	Offset (Bytes)
Length of the entire log record	int	0(4)
Type of log record (See Table 84 on page 593.)	short	4(2)
Log record general flag[1]	short	6(2)

Table 83. Log Manager Log Record Header (LogManagerLogRecordHeader) (continued)

Description	Type	Offset (Bytes)
Log Sequence Number of the previous log record written by this transaction. It is used to chain log records by transaction. If the value is 0000 0000 0000, this is the first log record written by the transaction.	SQLU_LSN[2]	8(6)
Unique transaction identifier	SQLU_TID[3]	14(6)
Log Sequence Number of the log record for this transaction prior to the log record being compensated. (Note: For compensation and backout free log records only.)	SQLU_LSN	20(6)
Log Sequence Number of the log record for this transaction being compensated. (Note: For propagatable compensation log records only.)	SQLU_LSN	26(6)
Total Length for Log Manager Log Record Header: • *Non Compensation: 20 bytes* • *Compensation: 26 bytes* • *Propagatable Compensation: 32 bytes*		

Notes:

1. Log record general flag constants

```
Redo Always                0x0001
Propagatable               0x0002
Conditionally Recoverable  0x0080
```

2. Log Sequence Number (LSN)

```
A unique log record identifier representing the relative byte address
of the log record within the database log.

SQLU_LSN: union {  char  [6] ;
                   short [3] ;
                 }
```

3. Transaction Identifier (TID)

```
A unique log record identifier representing the transaction.

SQLU_TID: union {  char  [6] ;
                   short [3] ;
                 }
```

Table 84. Log Manager Log Record Header Log Type Values and Definitions

Value	**Definition**
0x0061	Datalink manager log record
0x006F	Backup start
0x0041	Normal abort
0x004F	Backup end
0x0042	Backout free
0x0089	Table space roll forward to PIT starts
0x0063	MPP coordinator commit
0x0050	Table quiesce
0x0043	Compensation
0x0071	Table space roll forward to PIT ends
0x0044	Table space rolled forward
0x0051	Global pending list
0x0045	Local pending list
0x0052	Redo
0x0088	Forget transaction
0x0085	MPP subordinate commit
0x0080	MPP log synchronization
0x0053	Compensation required
G	Load pending list
0x0054	Partial abort
0x0048	Table load delete start
0x0055	Undo
0x0069	Propagate only
V	Migration begin
0x0049	Heuristic abort
0x0056	Migration end
0x004A	Load start
0x0083	TM prepare
0x004B	Load delete start compensation
0x0087	Heuristic commit
L	Lock description
0x0081	MPP prepare

Table 84. Log Manager Log Record Header Log Type Values and Definitions (continued)

Value	Definition
0x0084	Normal commit
0x0082	XA prepare
0x004E	Normal

Data Manager Log Records

Data manager log records are the result of DDL, DML, or Utility activities.

There are two types of data manager log records:
- Data Management System (DMS) logs have a component identifier of 1 in their header.
- Data Object Manager (DOM) logs have a component identifier of 4 in their header.

Table 85. DMS Log Record Header Structure (DMSLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=1)	unsigned char	0(1)
Function identifier (See Table 86.)	unsigned char	1(1)
Table identifiers		
Table space identifier	unsigned short	2(2)
Table identifier	unsigned short	4(2)
Total Length: 6 bytes		

Table 86. DMS Log Record Header Structure Function Identifier Values and Definitions

Value	Definition
102	Add columns to table
104	Undo add columns
106	Delete record
110	Undo insert record
111	Undo delete record
112	Undo update record
113	Add columns to table
104	Alter column length

Table 86. DMS Log Record Header Structure Function Identifier Values and Definitions (continued)

Value	Definition
115	Undo alter column length
118	Insert record
120	Update record
124	Alter table attribute
128	Initialize table
129	Delete record to empty page
130	Insert record to empty page
131	Undo insert record to empty page
132	Undo delete record to empty page

Table 87. DOM Log Record Header Structure (DOMLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=4)	unsigned char	0(1)
Function identifier (See Table 88.)	unsigned char	1(1)
Object identifiers		
Table space identifier	unsigned short	2(2)
Object identifier	unsigned short	4(2)
Table identifiers		
Table space identifier	unsigned short	6(2)
Table identifier	unsigned short	8(2)
Object type	unsigned char	10(1)
Flags	unsigned char	11(1)
Total Length: 12 bytes		

Table 88. DOM Log Record Header Structure Function Identifier Values and Definitions

Value	Definition
2	Create index
3	Drop index
4	Drop table
11	Truncate table (import replace)

Table 88. DOM Log Record Header Structure Function Identifier Values and Definitions (continued)

Value	Definition
35	Reorg table
101	Create table
130	Undo create table

Note: All data manager log record offsets are from the end of the log manager record header.

All log records whose function identifier short name begins with `UNDO` are log records written during the UNDO or ROLLBACK of the action in question.

The ROLLBACK can be a result of:

- The user issuing the ROLLBACK transaction statement
- A deadlock causing the ROLLBACK of a selected transaction
- The ROLLBACK of uncommitted transactions following a crash recovery
- The ROLLBACK of uncommitted transactions following a RESTORE and ROLLFORWARD of the logs.

Initialize Table

The initialize table log record is written when a new permanent table is being created; it signifies table initialization. This record appears after any log records that create the DATA storage object, and before any log records that create the LF and LOB storage objects. This is a Redo log record.

Table 89. Initialize Table Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
File create LSN	SQLU_LSN	6(6)
Table directory record	variable	12(72)
record type	unsigned char	12(1)
reserved	char	13(1)
index flag	unsigned short	14(2)
index root page	sqluint32	16(4)
TDESC recid	sqluint32	20(4)
reserved	char	24(56)
flags[1]	sqluint32	80(4)
Table description length	sqluint32	84(4)

Table 89. Initialize Table Log Record Structure (continued)

Description	Type	Offset (Bytes)
Table description record	variable	88(variable)
record type	unsigned char	88(1)
reserved	char	89(1)
number of columns	unsigned short	90(2)
array	variable long	92(variable)
Total Length: 88 bytes plus table description record length		

Notes:

1. Bit 0x00000010 indicates that the table was created with the VALUE COMPRESSION option. Bit 0x00000020 indicates that the table was created with the NOT LOGGED INITIALLY option, and that no DML activity on this table is logged until the transaction that created the table has been committed. Bit 0x00000800 indicates that the table was a mulitdimensional clustered (MDC) table created with the ORGANIZE BY clause.

Table Description Record: column descriptor array:

(number of columns) * 8, where each element of the array contains:

- field type (unsigned short, 2 bytes)

```
SMALLINT     0x0000
INTEGER      0x0001
DECIMAL      0x0002
DOUBLE       0x0003
REAL         0x0004
BIGINT       0x0005
CHAR         0x0100
VARCHAR      0x0101
LONG VARCHAR 0x0104
DATE         0x0105
TIME         0x0106
TIMESTAMP    0x0107
BLOB         0x0108
CLOB         0x0109
DATALINK     0x010E
GRAPHIC      0x0200
VARGRAPH     0x0201
LONG VARG    0x0202
DBCLOB       0x0203
```

- length (2 bytes)
  - If BLOB, CLOB, or DBCLOB, this field is not used. For the maximum length of this field, see the array that follows the column descriptor array.
  - If not DECIMAL, length is the maximum length of the field (short).

  - If PACKED DECIMAL: Byte 1, unsigned char, precision (total length) Byte 2, unsigned char, scale (fraction digits).
- null flag (unsigned short, 2 bytes)
  - mutually exclusive: allows nulls, or does not allow nulls
  - valid options: no default, type default, user default, or compress type default

```
ISNULL                  0x01
NONULLS                 0x02
TYPE_DEFAULT            0x04
USER_DEFAULT            0x08
COMPRESS_SYSTEM_DEFAULT 0x80
```

- field offset (unsigned short, 2 bytes) This is the offset from the start of the formatted record to where the field's fixed value can be found.

Table Description Record: LOB descriptor array:

(number of LOB, CLOB, and DBCLOB fields) * 12, where each element of the array contains:

- length (MAX LENGTH OF FIELD, sqluint32, 4 bytes)
- reserved (internal, sqluint32, 4 bytes)
- log flag (IS COLUMN LOGGED, sqluint32. 4 bytes)

The first LOB, CLOB, or DBCLOB encountered in the column descriptor array uses the first element in the LOB descriptor array. The second LOB, CLOB, or DBCLOB encountered in the column descriptor array uses the second element in the LOB descriptor array, and so on.

Import Replace (Truncate)

The import replace (truncate) log record is written when an IMPORT REPLACE action is being executed. This record indicates the re-initialization of the table (no user records, new life LSN). The second set of table space and object IDs in the log header identify the table being truncated (IMPORT REPLACE). This is a Redo log record.

Table 90. Import Replace (Truncate) Log Record Structure

Description	**Type**	**Offset (Bytes)**
Log header	DOMLogRecordHeader	0(12)
internal	variable	12(variable)
Total Length: 12 bytes plus variable length		

Rollback Insert

The rollback insert log record is written when an insert row action (INSERT RECORD) is rolled back. This is a Compensation log record.

Table 91. Rollback Insert Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
RID	sqluint32	8(4)
Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
Total Length: 16 bytes		

Reorg Table

The reorg table log record is written when the REORG utility has committed to completing the reorganization of a table. This is a Normal log record.

Table 92. Reorg Table Log Record Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(392)
Index token[1]	unsigned short	2(404)
Temporary table space ID[2]	unsigned short	2(406)
Total Length: 408 bytes		

Notes:

1. If not 0, it is the index by which the reorg is clustered (clustering index).
2. If not 0, it is the system temporary table space that was used to build the reorg.

Create Index, Drop Index

These log records are written when indexes are created or dropped. The two elements of the log record are:

- The index root page, which is an internal identifier
- The index token, which is equivalent to the `IID` column in SYSIBM.SYSINDEXES. If the value for this element is 0, the log record represents an action on an internal index, and is not related to any user index.

This is a normal log record.

Table 93. Create Index, Drop Index Log Records Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)

Table 93. Create Index, Drop Index Log Records Structure (continued)

Description	Type	Offset (Bytes)
Padding	char[ ]	12(2)
Index token	unsigned short	14(2)
Index root page	sqluint32	16(4)
Total Length: 20 bytes		

Create Table, Drop Table, Rollback Create Table, Rollback Drop Table

These log records are written when the DATA object for a permanent table is created or dropped. The DATA object is created during a CREATE TABLE, and prior to table initialization (Initialize Table). Create table and drop table are Normal log records. Rollback create table and rollback drop table are Compensation log records.

Table 94. Create Table, Drop Table, Rollback Create Table, Rollback Drop Table Log Records Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(72)
Total Length: 84 bytes		

Alter Table Attribute

The alter table attribute log record is written when the state of a table is changed VIA the ALTER TABLE statement or as a result of adding or validating constraints.

Table 95. Alter Table Attribute, Undo Alter Table Attribute

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
Alter bit (attribute) mask	sqluint32	8(4)
Alter bit (attribute) values	sqluint32	12(4)
Total Length: 16 bytes		

Attribute Bits:

```
0x00000001   Propagation
0x00000002   Check Pending
0x00000010   Value Compression
0x00010000   Append Mode
0x00200000   LF Propagation
```

If one of the bits above is present in the alter bit mask, then this attribute of the table is being altered. To determine the new value of the table attribute (0 = OFF and 1 = ON), check the corresponding bit in the alter bit value.

Alter Table Add Columns, Rollback Add Columns

The alter table add columns log record is written when the user is adding columns to an existing table using an ALTER TABLE statement. Complete information on the old columns and new columns is logged.

- Column count elements represent the old number of columns and the new total number of columns.
- The parallel arrays contain information about the columns defined in the table. The old parallel array defines the table prior to the ALTER TABLE statement, while the new parallel array defines the table resulting from ALTER TABLE statement.
- Each parallel array consists of:
  - An array equivalent to the column descriptor array in the table description record (see "Initialize Table" on page 596).
  - A second array equivalent to the LOB descriptor array in the table description record. However, since this array is parallel to the first, the only elements used are those whose corresponding element in the first array are of type BLOB, CLOB, or DBCLOB.

Alter table add columns is a Normal log record. Rollback add columns is a Compensation log record.

Table 96. Alter Table Add Columns, Rollback Add Columns Log Records Structure

Description	**Type**	**Offset (Bytes)**
Log header	DMSLogRecordheader	0(6)
Padding	char[ ]	6(2)
Old column count	sqluint32	8(4)
New column count	sqluint32	12(4)
Old parallel arrays[1]	variable	16(variable)
New parallel arrays[2]	variable	variable
*Total Length: 40 bytes plus 2 sets of parallel arrays; array size is (old/new column count) * 20.*		

Array Elements:

1. Each element in this array is 8 bytes long.
2. Each element in this array is 12 bytes long.

For information about the column descriptor array or the LOB descriptor array, see Table 89 on page 596).

Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record

These log records are written when rows are inserted into or deleted from a table. Insert record and delete record log records are generated during an update if the location of the record being updated must be changed to accommodate the modified record data. Insert record and delete record are Normal log records. Rollback delete record and rollback update record are Compensation log records.

Table 97. Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record Log Records Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
RID	sqluint32	8(4)
Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
Record offset	unsigned short	16(2)
Record header and data	variable	18(variable)
Total Length: 18 bytes plus Record length		

Record Header and Data Details:

Record header

- 4 bytes
- Record type[a] (unsigned char, 1 byte).
  - Bit values represent different classes and possible types within the classes. Records are one of two classes:
    - Updatable
    - Special control
  - Each class can contain the three types:
    - Normal
    - Pointer
    - Overflow
  - The record contains user data if
    - the record type is 0x00 or 0x10
    - the bit 0x04 is set.
- Reserved (char, 1 byte)
- Record length (unsigned short, 2 bytes)

Record

- variable length
- Record type (unsigned char, 1 byte). Updatable records are one of three types:
  - 0 - Internal control
  - 1 - Formatted user data without VALUE COMPRESSION option
  - 2 - Formatted user data with VALUE COMPRESSION option
- Reserved (char, 1 byte)
- The rest of the record is dependent upon the record type and the table descriptor record defined for the table. If the record type is an internal control, the data cannot be viewed.
- The following fields apply to user data records with record type value 1
  - Fixed length (unsigned short, 2 bytes). This is the length of the fixed length section of the data row.
  - Formatted record (fixed and variable length).
- The following fields apply to user data records with record type value 2
  - Number of columns (unsigned short, 2 bytes). This is the number of columns in the data portion of the data row.
  - Formatted record (offset array and data portion).

[a] Record data can only be viewed if the record type (specified in the record header) is updatable (that is, *not* special control).

Formatted User Data Record for table without VALUE COMPRESSION

The formatted record of a table created/altered without the VALUE COMPRESSION can be a combination of fixed and variable length data. All fields contain a fixed length portion. In addition, there are eight field types that have variable length parts:

- VARCHAR
- LONG VARCHAR
- DATALINK
- BLOB
- CLOB
- VARGRAPHIC
- LONG VARG
- DBCLOB

The length of the fixed portion of the different field types can be determined as follows:

- DECIMAL

  This field is a standard packed decimal in the form: *nnnnnn...s*. The length of the field is: (precision + 2)/2. The sign nibble (s) is xC for positive (+), and xD or xB for negative (−).
- SMALLINT INTEGER BIGINT DOUBLE REAL CHAR GRAPHIC

  The length field in the element for this column in the table descriptor record contains the fixed length size of the field.
- DATE

  This field is a 4-byte packed decimal in the form: *yyyymmdd*. For example, April 3, 1996 is represented as x'19960403'.
- TIME

  This field is a 3-byte packed decimal in the form: *hhmmss*. For example, 1:32PM is represented as x'133200'.
- TIMESTAMP

  This field is a 10-byte packed decimal in the form: *yyyymmddhhmmssuuuuuu* (DATE|TIME|microseconds).
- VARCHAR LONG VARCHAR DATALINK BLOB CLOB VARGRAPHIC LONG VARG DBCLOB

  The length of the fixed portion of all the variable length fields is 4.

Note: For element addresses, see Table 89 on page 596.

The following sections describe the location of the fixed portion of each field within the formatted record.

The table descriptor record describes the column format of the table. It contains an array of column structures, whose elements represent field type, field length, null flag, and field offset. The latter is the offset from the beginning of the formatted record, where the fixed length portion of the field is located.

Table 98. Table Descriptor Record Structure

record type	number of columns	column structure • field type • length • null flag • field offset	LOB information

Note: For more information, see Table 89 on page 596.

For columns that are nullable (as specified by the null flag), there is an additional byte following the fixed length portion of the field. This byte contains one of two values:

- NOT NULL (0x00)
- NULL (0x01)

If the null flag within the formatted record for a column that is nullable is set to 0x00, there is a valid value in the fixed length data portion of the record. If the null flag value is 0x01, the data field value is NULL.

The formatted user data record contains the table data that is visible to the user. It is formatted as a fixed length record, followed by a variable length section.

Table 99. Formatted User Data Record Structure for table without VALUE COMPRESSION

record type	length of fixed section	fixed length section	variable data section

Note: For more information, see Table 97 on page 602.

All variable field types have a 4-byte fixed data portion in the fixed length section (plus a null flag, if the column is nullable). The first 2 bytes (short) represent the offset from the beginning of the fixed length section, where the variable data is located. The next 2 bytes (short) specify the length of the variable data referenced by the offset value.

Formatted User Data Record for table with VALUE COMPRESSION

The formatted record for a table created or altered with VALUE COMPRESSION consists of the offset array and the data portion. Each entry in the array is a 2-byte offset to the corresponding column data in the data portion. The number of column data in the data portion can be found in the record header and the number of entries in the offset array is one plus the number of column data that exists in the data portion.

1. Compressed column values consumes only one byte of disk space which is used for attribute byte. The attribute byte will indicate the column data is compressed, for example, the data value is known but is not stored on disk. The high bit (0x80) in the offset will be used to indicate the accessed data is an attribute byte. (So only 15 bits are used to represent the offset of the corresponding column data.)
2. For regular column data, the column data follows. There will not be any attribute byte nor any length indicator present.
3. Accessed data can take on two different values if it is an attribute byte:
   - `NULL   0x01   (Value is NULL)`

- `COMPRESSED SYSTEM DEFAULT   0x80   (Value is equal to the system default)`

4. The length of column data is the difference between the current offset and the offset of the next column.

Table 100. Formatted User Data Record Structure for table with VALUE COMPRESSION

record type	number of column in data portion	offset array	data portion

Note: For more information, see Table 97 on page 602.

Insert Record to Empty Page, Delete Record to Empty Page, Rollback Delete Record to Empty Page, Rollback Insert Record to Empty Page

These log records are written when the table is a multidimensional clustered (MDC) table. The Insert To Empty Page log record is written when a record is inserted and it is the first record on a page, where that page is not the first page of a block. This log record logs the insert to the page, as well as the update of a bit on the first page of the block, indicating that that page is no longer empty. The Delete To Empty Page log record is written when the last record is deleted from a page, where that page is not the first page of a block. This log record logs the delete from the page, as well as the update of a bit on the first page of the block, indicating that the page is empty. Insert record and Delete record to Empty Page are Normal log records. Rollback delete record and rollback insert record are Compensation log records.

Table 101. Insert Record to Empty Page, Delete Record to Empty Page, Rollback Delete Record to Empty Page, Rollback Insert Record to Empty Page

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[]	6(2)
RID	sqluint32	8(4)
Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
First page of the block	sqluint32	16(4)
Record offset	unsigned short	20(2)
Record header and data	variable	22(variable)
Total Length: 22 bytes plus Record length		

Note: For Record Header and Data Details, see Table 97 on page 602.

Update Record

The update record log record is written when a row is updated, and if its storage location does not change. There are two available log record formats; they are identical to the insert record and the delete record log records (see "Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record" on page 602). One contains the *pre*-update image of the row being updated; the other contains the *post*-update image of the row being updated. This is a Normal log record.

Table 102. Update Record Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
RID	sqluint32	8(4)
New Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
Record offset	unsigned short	16(2)
Old record header and data	variable	18(variable)
Log header	DMSLogRecordHeader	variable(6)
Padding	char[ ]	variable(2)
RID	sqluint32	variable(4)
Old record length	unsigned short	variable(2)
Free space	unsigned short	variable(2)
Record offset	unsigned short	variable(2)
New record header and data	variable	variable(variable)
Total Length: 36 bytes plus 2 Record lengths		

Long Field Manager Log Records

Long field manager log records are written only if a database is configured with LOG RETAIN on or USEREXITS enabled. They are written whenever long field data is inserted, deleted, or updated.

To conserve log space, long field data inserted into tables is not logged if the database is configured for circular logging. In addition, when a long field value is updated, the before image is shadowed and not logged.

All long field manager log records begin with a header.

Long Field Manager Log Records

All long field manager log record offsets are from the end of the log manager log record header.

When a table has been altered to capture LONG VARCHAR OR LONG VARGRAPHIC columns (by specifying INCLUDE LONGVAR COLUMNS on the ALTER TABLE statement):

- The long field manager will write the appropriate long field log record.
- When long field data is updated, the update is treated as a delete of the old long field value, followed by an insert of the new value. To determine whether or not a Delete/Add Long Field Record is associated with an update operation on the table the original operation value would be logged to the Long Field Manager Log Record.
- When tables with long field columns are updated, but the long field columns themselves are not updated, a Non-update Long Field Record is written.
- The Delete Long Field Record and the Non-update Long Field Record are information only log records.

Table 103. Long Field Manager Log Record Header (LongFieldLogRecordHeader)

Description	Type	Offset (Bytes)
Originator code (component identifier = 3)	unsigned char	0(1)
Operation type (See Table 104.)	unsigned char	1(1)
Table space identifier	unsigned short	2(2)
Object identifier	unsigned short	4(2)
Parent table space identifier[1]	unsigned short	6(2)
Parent object identifier[2]	unsigned short	8(2)
Total Length: 10 bytes		

Notes:

1. Table space ID of the data object
2. Object ID of the data object

Table 104. Long Field Manager Log Record Header Operation Type Values and Definitions

Value	Definition
113	Add Long Field Record
114	Delete Long Field Record
115	Non-Update Long Field Record

Add/Delete/Non-update Long Field Record

These log records are written whenever long field data is inserted, deleted, or updated. The length of the data is rounded up to the next 512-byte boundary.

Table 105. Add/Delete/Non-update Long Field Record Log Record Structure

Description	Type	Offset (Bytes)
Log header	LongFieldLogRecordHeader	0(10)
Reserved	char	10(1)
Original operation type[1]	char	11(1)
Column identifier[2]	char	12(2)
Long field length[3]	unsigned short	14(2)
File offset[4]	sqluint32	16(4)
Long field data	char[ ]	20(variable)

Notes:

1. Original operation type

```
1     Insert
2     Delete
4     Update
```

2. The column number that the log record is applied to. Column number starts from 0.
3. Long field data length in 512-byte sectors (actual data length is not logged). The value of this field is always positive. The long field manager never writes log records for zero length long field data that is being inserted, deleted, or updated.
4. 512-byte sector offset into long field object where data is to be located.

Transaction Manager Log Records

The transaction manager produces log records signifying the completion of transaction events (for example, commit or rollback). The time stamps in the log records are in Coordinated Universal Time (CUT), and mark the time (in seconds) since January 01, 1970.

Normal Commit

This log record is written for XA transactions in a single-node environment, or on the coordinator node in MPP. It is only used for XA applications. The log record is written when a transaction commits after one of the following events:

- A user has issued a COMMIT
- An implicit commit occurs during a CONNECT RESET.

Table 106. Normal Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
Authorization identifier of the application (if the log record is marked as propagatable)	char [ ]	28(variable)
Total Length: 28 bytes plus variable propagatable (28 bytes non-propagatable)		

Heuristic Commit

This log record is written when an indoubt transaction is committed.

Table 107. Heuristic Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
Authorization identifier of the application (if the log record is marked as propagatable)	char [ ]	28(variable)
Total Length: 28 bytes plus variable propagatable (28 bytes non-propagatable)		

MPP Coordinator Commit

This log record is written on a coordinator node for an application that performs updates on at least one subordinator node.

Table 108. MPP Coordinator Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
MPP identifier of the transaction	SQLP_GXID	28(20)
Maximum node number	unsigned short	48(2)
TNL	unsigned char [ ]	50(max node number/8 + 1)
Authorization identifier of the application (if the log record is marked as propagatable)	char [ ]	variable(variable)
Total Length: variable		

MPP Subordinator Commit

This log record is written on a subordinator node in MPP.

Table 109. MPP Subordinator Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
MPP identifier of the transaction	SQLP_GXID	28(20)
Reserved	unsigned short	48(variable + 2)
Authorization identifier (if the log record is marked as propagatable)	char [ ]	variable(variable)
Total Length: 48 bytes plus variable propagatable (48 bytes non-propagatable)		

Normal Abort

This log record is written when a transaction aborts after one of the following events:

- A user has issued a ROLLBACK
- A deadlock occurs
- An implicit rollback occurs during crash recovery
- An implicit rollback occurs during ROLLFORWARD recovery.

Table 110. Normal Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier of the application (if the log record is marked as propagatable)	char [ ]	20(variable)
Total Length: 20 bytes plus variable (20 bytes non-propagatable)		

Heuristic Abort

This log record is written when an indoubt transaction is aborted.

Table 111. Heuristic Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier of the application (if the log record is marked as propagatable)	char [ ]	20(variable)
Total Length: 20 bytes plus variable (20 bytes non-propagatable)		

Local Pending List

This log record is written if a transaction commits and a pending list exists. The pending list is a linked list of non-recoverable operations (such as deletion of a file) that can only be performed when the user/application issues a COMMIT. The variable length structure contains the pending list entries.

Table 112. Local Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
Authorization identifier length[1]	unsigned short	28(2)
Authorization identifier of the application[1]	char [ ]	30(variable)[2]
Pending list entries	variable	variable(variable)
Total Length: 30 bytes plus variables propagatable (28 bytes plus pending list entries non-propagatable)		

Notes:

1. If the log record is marked as propagatable
2. Variable based on Authorization identifier length

Global Pending List

This log record is written if a transaction involved in a two-phase commit commits, and a pending list exists. The pending list contains non-recoverable operations (such as deletion of a file) that can only be performed when the user/application issues a COMMIT. The variable length structure contains the pending list entries.

Table 113. Global Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier length[1]	unsigned short	20(2)
Authorization identifier of the application[1]	char [ ]	22(variable)[2]
Global pending list entries	variable	variable(variable)
Total Length: 22 bytes plus variables propagatable (20 bytes plus pending list entries non-propagatable)		

Notes:

1. If the log record is marked as propagatable
2. Variable based on Authorization identifier length

XA Prepare

This log record is written for XA transactions in a single-node environment, or on the coordinator node in MPP. It is only used for XA applications. The log record is written to mark the preparation of the transaction as part of a two-phase commit. The XA prepare log record describes the application that started the transaction, and is used to recreate an indoubt transaction.

Table 114. XA Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction prepared	sqluint64	20(8)
Log space used by transaction	sqluint64	28(8)
Transaction Node List Size	sqluint32	36(4)
Transaction Node List	unsigned char [ ]	40(variable)
XA identifier of the transaction	SQLXA_XID	variable(140)
Application Information Length	sqluint32	variable(4)
Code Page Identifier	sqluint32	variable(4)
Transaction Start Time	sqluint32	variable(4)
Application name	char [ ]	variable(20)
Application identifier	char [ ]	variable(32)
Sequence number	char [ ]	variable(4)
Database alias used by client	char [ ]	240(20)
Authorization identifier	char [ ]	variable(variable)
Synclog information	variable	variable(variable)
Total Length: 268 bytes plus variables		

MPP Subordinator Prepare

This log record is written for MPP transactions on subordinator nodes. The log record is written to mark the preparation of the transaction as part of a two-phase commit. The MPP subordinator prepare log record describes the application that started the transaction, and is used to recreate an indoubt transaction.

Table 115. MPP Subordinator Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time Transaction Prepared	sqluint64	20(8)
Log space used by transaction	sqluint64	28(8)
Coordinator LSN	SQLP_LSN	36(6)
Padding	char [ ]	42(2)

Table 115. MPP Subordinator Prepare Log Record Structure (continued)

Description	Type	Offset (Bytes)
MPP identifier of the transaction	SQLP_GXID	44(20)
Application Information Length	sqluint32	64(4)
Code page	sqluint32	68(4)
Transaction Start Time	sqluint32	72(4)
Application name	char [ ]	76(20)
Application identifier	char [ ]	96(32)
Sequence number	char [ ]	128(4)
Database alias used by client	char [ ]	132(20)
Authorization identifier	char [ ]	152(variable)
Total Length: 152 bytes plus variable		

Backout Free

This log record is used to mark the end of a backout free interval. The backout free interval is a set of log records that is not to be compensated if the transaction aborts. This log record contains only a 6-byte log sequence number (*complsn*, stored in the log record header starting at offset 20). When this log record is read during rollback (following an aborted transaction), *complsn* marks the next log record to be compensated.

Table 116. Migration Begin Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Complsn	SQLP_LSN	20(6)
Total Length: 26 bytes		

Utility Manager Log Records

The utility manager produces log records associated with the following DB2 UDB utilities:

- Migration
- Load
- Backup
- Table space rollforward.

The log records signify the beginning or the end of the requested activity. All utility manager log records are marked as propagatable regardless of the tables that they affect.

Migration Begin

This log record is associated with the beginning of catalog migration.

Table 117. Migration Begin Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Migration start time	char[ ]	20(10)
Migrate from release	unsigned short	30(2)
Migrate to release	unsigned short	32(2)
Total Length: 34 bytes		

Migration End

This log record is associated with the successful completion of catalog migration.

Table 118. Migration End Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Migration end time	char[ ]	20(10)
Migrate to release	unsigned short	30(2)
Total Length: 32 bytes		

Load Start

This log record is associated with the beginning of a load.

Table 119. Load Start Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Log record identifier	sqluint32	20(4)
Pool identifier	unsigned short	24(2)
Object identifier	unsigned short	26(2)
Flag	unsigned char	28(1)
Object pool list	variable	29(variable)
Total Length: 29 bytes plus variable		

Table Load Delete Start

This log record is associated with the beginning of the delete phase in a load operation. The delete phase is started only if there are duplicate primary key values.

Table 120. Table Load Delete Start Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Total Length: 20 bytes		

Load Delete Start Compensation

This log record is associated with the end of the delete phase in a load operation.

Table 121. Load Delete Start Compensation Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Total Length: 20 bytes		

Load Pending List

This log record is written when a load transaction commits. The pending list is a linked list of non-recoverable operations which are deferred until the transaction commits. No commit log record follows this transaction.

Table 122. Load Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
Authorization identifier of the application (if the log record is marked as propagatable)	char[ ]	28(9)
Pending list entries	variable	37(variable)
Total Length: 37 bytes plus pending list entries propagatable (28 bytes plus pending list entries non-propagatable)		

Backup End

This log record is associated with the end of a successful backup.

Table 123. Backup End Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Backup end time	sqluint64	20(8)
Total Length: 28 bytes		

Table Space Rolled Forward

This log record is associated with table space ROLLFORWARD recovery. It is written for each table space that is successfully rolled forward.

Table 124. Table Space Rolled Forward Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Table space identifier	unsigned short	20(2)
Total Length: 22 bytes		

Table Space Roll Forward to PIT Begins

This log record is associated with table space ROLLFORWARD recovery. It marks the beginning of a table space roll forward to a point in time.

Table 125. Table Space Roll Forward to PIT Begins Log Record Structure

Description	Type	Offset (Bytes)
Time stamp for this log record.	sqluint64	0(8)
Time stamp to which table spaces are being rolled forward.	sqluint64	8(8)
Number of pools being rolled forward.	unsigned short	16(2)
Integer list of pool IDs that are being rolled forward.	int*numpools	18(variable)
Total Length: 10 bytes plus variable		

Table Space Roll Forward to PIT Ends

This log record is associated with table space ROLLFORWARD recovery. It marks the end of a table space roll forward to a point in time.

Table 126. Table Space Roll Forward to PIT Ends Log Record Structure

Description	Type	Offset (Bytes)
Time stamp for this log record.	sqluint64	0(8)
Time stamp to which table spaces were rolled forward.	sqluint32	8(8)
A flag whose value is `TRUE` if the roll forward was successful, or `FALSE` if the roll forward was canceled.	int	16(4)
Total Length: 24 bytes		

Datalink Manager Log Records

Datalink manager log records are the result of DDL, DML, or completion of transaction events involving DATALINK columns. These log records are written only when the DDL or the DML involves DATALINK columns with the file link control attribute.

Table 127. Datalink Manager Log Record Header Structure (DLMLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=8)	unsigned char	0(1)
Function identifier (See Table 128.)	unsigned char	1(1)
padding	char []	2(2)
Total Length: 6 bytes		

Table 128. Datalink Manager Log Record Header Function Identifiers and Values

Identifier	Value
LINK_FILE	33
UNLINK_FILE	34
DELETE_GROUP	35
DELETE_PGROUP	36
DLFM_PREPARE	37

Link File

The link file log record is written when an insert or an update on a table with a DATALINK column creates a link to a file. One log record is written for each new link that is created. This log record is only used for undo.

Table 129. Link File Log Record Structure

Description	Type	Offset (Bytes)
Log header	DLMLogRecordHeader	0(4)
ServerId	sqlint32	4(4)
ReadOnly	int	8(4)
AuthId	char []	12(8)
GroupId	char []	20(17)
Operation Type (See Table 130 on page 619.)	char []	37(1)
AccessControl	unsigned short	38(2)
PrefixId	char []	40(9)

Table 129. Link File Log Record Structure (continued)

Description	Type	Offset (Bytes)
padding	char []	49(3)
RecoveryId	char []	52(7)
padding	char []	59(1)
Time stamp	sqluint32	60(4)
StemNameLen	sqluint32	64(4)
StemName	variable	68(variable)
ServerNameLen[1]	sqluint32	variable(4)
PrefixNameLen[1]	sqluint32	variable(4)
ServeNamePrefixName[1]	variable	variable(variable)
Total Length: 68 plus StemNameLen if non-progagatable (76 plus StemNameLen plus ServerNameLen plus PrefixNameLen if propagatable)		

Notes:

1. If the log record is propagatable.

Table 130. Link File Log Record Structure Operation Types and Values

Identifier	Value
LINK_FILE_ONLY (value constructed by DLVALUE)	0
LINK_NEW_VERSION (value constructed by DLNEWCOPY)	10
LINK_PREVIOUS_VERSION (value constructed by DLPREVIOUSCOPY)	20
LINK_REPLACE_CONTENT (value constructed by DLREPLACECONTENT)	30

Unlink File

The unlink file log record is written when a delete or an update on a table with a DATALINK column drops a link to a file. One log record is written for each link that is dropped. This log record is only used for undo.

Table 131. Unlink File Log Record Structure

Description	Type	Offset (Bytes)
Log header	DLMLogRecordHeader	0(4)
ServerId	sqlint32	4(4)
PrefixId	char []	8(9)

Table 131. Unlink File Log Record Structure (continued)

Description	Type	Offset (Bytes)
Operation type (See Table 132.)	char []	17(1)
padding	char []	18(2)
RecoveryId	char []	20(7)
padding	char []	27(1)
Time stamp	sqluint32	28(4)
StemNameLen	sqluint32	32(4)
StemName	variable	36(variable)
poolID[1]	unsigned short	variable(2)
objectID[1]	unsigned short	variable(2)
colNum[1]	unsigned short	variable(2)
padding[1]	char []	variable(2)
ServerNameLen[1]	sqluint32	variable(4)
PrefixNameLen[1]	sqluint32	variable(4)
ServerNamePrefixName[1]	variable	variable(variable)
Total Length: 36 plus StemNameLen if non-propagatable (52 plus StemNameLen plus ServerNameLen plus PrefixNameLen if propagatable)		

Notes:

1. If the log record is propagatable.

Table 132. Link File Log Record Structure Operation Types and Values

Identifier	Value
UNLINK_REGULAR_FILE	0
UNLINK_UPDATE_IN_PLACE_FILE	10

Delete Group

The delete group log record is written when a table with DATALINK columns (having the file link control attribute) is dropped. One log record is written for each such DATALINK column for each DB2 Data Links Manager configured to the database. For a given DB2 Data Links Manager, the log record is written only if that DB2 Data Links Manager has the group defined on it when the table is dropped. This log record is only used for undo.

Table 133. Delete Group Log Record Structure

Description	Type	Offset (Bytes)
Log header	DLMLogRecordHeader	0(4)
ServerId	sqlint32	4(4)
RecoveryId	char []	8(7)
padding	char []	15(1)
GroupId	char []	16(17)
padding	char []	33(3)
Total Length: 36 bytes		

Delete PGroup

The delete pgroup log record is written when a table space is dropped. One log record is written for each DB2 Data Links Manager configured to the database. For a given DB2 Data Links Manager, the log record is written only if that DB2 Data Links Manager has the pgroup defined on it when the table space is dropped. This log record is only used for undo.

Table 134. Delete PGroup Log Record Structure

Description	Type	Offset (Bytes)
Log header	DLMLogRecordHeader	0(4)
ServerId	sqlint32	4(4)
poolLifeLSN	SQLU_LSN	8(6)
poolId	unsigned short	14(2)
RecoveryId	char []	16(7)
padding	char []	23(1)
Total Length: 24 bytes		

DLFM Prepare

The DLFM prepare log record is written during the prepare phase, when a two-phase commit is used for transactions involving DB2 Data Links Manager. It is used to recreate a transaction for DB2 Data Links Managers that are in-doubt.

Table 135. DLFM Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	DLMLogRecordHeader	0(4)
NumDLFMs	unsigned short	4(4)
ServerIds	variable	8(variable)

Table 135. DLFM Prepare Log Record Structure (continued)

Description	Type	Offset (Bytes)
*Total Length: 8 bytes plus (NumDLFMs * 4)*		

Related reference:

- "db2ReadLog - Asynchronous Read Log" on page 185

Appendix G. Application Migration

Application Migration Considerations

This section describes issues that should be considered before migrating an application to Version 8.

There are four possible operating scenarios:
1. Running pre-Version 8 applications against databases that have not been migrated
2. Running pre-Version 8 applications against migrated databases
3. Updating applications with Version 8 APIs
4. Running Version 8 applications against migrated databases.

The first and the fourth are consistent operating environments that do not require qualification.

The second, in which only the databases have been migrated, should work without changes to any application, because back-level applications are supported. However, as with any new version, a small number of incompatibilities can occur.

For the third scenario, in which applications are to be updated with Version 8 APIs, the following points should be considered:
- All pre-Version 8 APIs that have been discontinued in Version 8 are still defined in the Version 8 header files, so that older applications will compile and link with Version 8 headers.
- Discontinued APIs should be removed from applications as soon as possible to enable these applications to take full advantage of the new functions available in Version 8, and to position the applications for future enhancements.
- The names of the APIs listed below have changed because of new function in Version 8. Users should scan for these names in their application source code to identify the changes required following Version 8 migration of the application.

  APIs that are not listed do not require changes following migration of an application.

  Note that an application may contain the generic version of an API call, depending on the application programming language being used. In all cases, the generic version of the API name is identical to the C version of the name, with the exception that the fourth character is always **g**.

Changed APIs and Data Structures

Table 136. Back-level Supported APIs and Data Structures

API or Data Structure (Version)	Descriptive Name	New API or Data Structure (Version)
sqlbftsq (V2)	Fetch Table Space Query	sqlbftpq (V5)
sqlbstsq (V2)	Single Table Space Query	sqlbstpq (V5)
sqlbtsq (V2)	Table Space Query	sqlbmtsq (V5)
sqlectdd (V2)	Catalog Database	sqlecadb (V5)
sqlepstart (V5)	Start Database Manager	db2InstanceStart (V8)
sqlepstp (V5)	Stop Database Manager	db2InstanceStop (V8)
sqlepstr (V2)	Start Database Manager (DB2 Parallel Edition Version 1.2)	db2InstanceStart (V8)
sqlestar (V2)	Start Database Manager (DB2 Version 2)	db2InstanceStart (V8)
sqlestop (V2)	Stop Database Manager	db2InstanceStop (V8)
sqlerstd (V5)	Restart Database	db2DatabaseRestart (V6)
sqlfddb (V7)	Get Database Configuration Defaults	db2CfgGet (V8)
sqlfdsys (V7)	Get Database Manager Configuration Defaults	db2CfgGet (V8)
sqlfrdb (V7)	Reset Database Configuration	db2CfgSet (V8)
sqlfrsys (V7)	Reset Database Manager Configuration	db2CfgSet (V8)
sqlfudb (V7)	Update Database Configuration	db2CfgSet (V8)
sqlfusys (V7)	Update Database Manager Configuration	db2CfgSet (V8)
sqlfxdb (V7)	Get Database Configuration	db2CfgGet (V8)
sqlfxsys (V7)	Get Database Configuration	db2CfgGet (V8)
sqlmon (V6)	Get/Update Monitor Switches	db2MonitorSwitches (V7)
sqlmonss (V5)	Get Snapshot	db2GetSnapshot (V6)
sqlmonsz (V6)	Estimate Size Required for sqlmonss() Output Buffer	db2GetSnapshotSize (V7)
sqlmrset (V6)	Reset Monitor	db2ResetMonitor (V7)
sqlubkp (V5)	Backup Database	db2Backup (V8)
sqlubkup (V2)	Backup Database	db2Backup (V8)
sqlugrpi (V2)	Get Row Partitioning Information (DB2 Parallel Edition Version 1.x)	sqlugrpn (V5)
sqluhcls (V5)	Close Recovery History File Scan	db2HistoryCloseScan (V6)
sqluhget (V5)	Retrieve DDL Information From the History File	db2HistoryGetEntry (V6)

Table 136. Back-level Supported APIs and Data Structures (continued)

API or Data Structure (Version)	Descriptive Name	New API or Data Structure (Version)
sqluhgne (V5)	Get Next Recovery History File Entry	db2HistoryGetEntry (V6)
sqluhops (V5)	Open Recovery History File Scan	db2HistoryOpenScan (V6)
sqluhprn (V5)	Prune Recovery History File	db2Prune (V6)
sqluhupd (V5)	Update Recovery History File	db2HistoryUpdate (V6)
sqluload (V7)	Load	db2Load (V8)
sqluqry (V5)	Load Query	db2LoadQuery (V6)
sqlureot (V7)	Reorganize Table	db2Reorg (V8)
sqlurestore (V7)	Restore Database	db2Restore (V8)
sqlurlog (V7)	Asynchronous Read Log	db2ReadLog (V8)
sqluroll (V7)	Rollforward Database	db2Rollforward (V8)
sqlursto (V2)	Restore Database	sqlurst (V5)
sqlustat (V7)	Runstats	db2Runstats (V8)
sqlxhcom (V2)	Commit an Indoubt Transaction	sqlxphcm (V5)
sqlxhqry (V2)	List Indoubt Transactions	sqlxphqr (V5)
sqlxhrol (V2)	Roll Back an Indoubt Transaction	sqlxphrl (V5)
sqlxphqr (V7)	List an Indoubt Transaction	db2XaListIndTrans (V8)
SQLB-TBSQRY-DATA (V2)	Table space data structure.	SQLB-TBSPQRY-DATA (V5)
SQLE-START-OPTIONS (V7)	Start Database Manager data structure	db2StartOptionsStruct (V8)
SQLEDBSTOPOPT (V7)	Start Database Manager data structure	db2StopOptionsStruct (V8)
SQLEDBSTRTOPT (V2)	Start Database Manager data structure (DB2 Parallel Edition Version 1.2)	db2StartOptionsStruct (V8)
SQLUHINFO and SQLUHADM (V5)	History file data structures.	db2HistData (V6)
SQLULOAD-IN (V7)	Load input structure	db2LoadIn (V8)
SQLULOAD-OUT (V7)	Load output structure	db2LoadOut (V8)
SQLXA-RECOVER (V7)	Transaction API structure	db2XaRecoverStruct

Table 137. Back-level Unsupported APIs

Name	Descriptive Name	APIs Supported in V8
sqlufrol/sqlgfrol	Roll Forward Database (DB2 Version 1.1)	db2Rollforward
sqluprfw	Rollforward Database (DB2 Parallel Edition Version 1.x)	db2Rollforward
sqlurfwd/sqlgrfwd	Roll Forward Database (DB2 Version 1.2)	db2Rollforward
sqlurllf/sqlgrfwd	Rollforward Database (DB2 Version 2)	db2Rollforward

Appendix H. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

```
htmlcdpath/doc/htmlcd/%L/category
```

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, `en_US`.
- *category* is the category identifier. For example, `core` for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d1e80` identifies the English version of the *Administration Guide: Planning* and the file name `db2d1g80` identifies the German version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	**Identifier**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category cover DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/core`.

Table 138. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is `doc/htmlcd/%L/admin`.

Table 139. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80

Table 139. Administration information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is `doc/htmlcd/%L/ad`.

Table 140. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is `doc/htmlcd/%L/wareh`.

Table 141. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is `doc/htmlcd/%L/conn`.

Table 142. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is `doc/htmlcd/%L/start`.

Table 143. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is `doc/htmlcd/%L/tutr`.

Table 144. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is `doc/htmlcd/%L/opt`.

Table 145. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80

Table 145. Optional component information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 146. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	doc/prodcd/*%L*/db2ir where *%L* is the language identifier.
DB2 Connect Release Notes	See note.	See note.	doc/prodcd/*%L*/db2cr where *%L* is the language identifier.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes is available from the Information Center and on the product CD-ROMs. To view the ASCII file:

- On UNIX-based platforms, see the `Release.Notes` file. This file is located in the `DB2DIR/Readme/%L` directory, where `%L` represents the locale name and `DB2DIR` represents:
  - `/usr/opt/db2_08_01` on AIX
  - `/opt/IBM/db2/V8.1` on all other UNIX operating systems
- On other platforms, see the `RELEASE.TXT` file. This file is located in the directory where the product is installed.

Related tasks:

- "Printing DB2 books from PDF files" on page 635

- "Ordering printed DB2 books" on page 636
- "Accessing online help" on page 636
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 640
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 641

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     `x:\doc\language` directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     `/cdrom/doc/%L` directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

Related tasks:

- "Ordering printed DB2 books" on page 636
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 640
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 641

Related reference:

- “Overview of DB2 Universal Database technical information” on page 627

Ordering printed DB2 books

Procedure:

To order printed books:

- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/shop/planetwide
- Phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

Related tasks:

- “Printing DB2 books from PDF files” on page 635
- “Finding topics by accessing the DB2 Information Center from a browser” on page 638
- “Viewing technical documentation online directly from the DB2 HTML Documentation CD” on page 641

Related reference:

- “Overview of DB2 Universal Database technical information” on page 627

Accessing online help

The online help that comes with all DB2 components is available in three types:

- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:

- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:

- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:

    ```
    ? command
    ```

    where *command* represents a keyword or the entire command.

  For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
- For Message help:

  ```
  ? XXXnnnnn
  ```

  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.
- For SQL statement help, open the command line processor and enter:
  - For SQL help:

    ```
    ? sqlstate or ? class code
    ```

    where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

    For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.
  - For SQLSTATE help:

```
help statement
```

where *statement* represents an SQL statement.

For example, `help SELECT` displays help about the SELECT statement.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 638
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 641

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:

- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the “Index” category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.
3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:

- "Accessibility" on page 647
- "DB2 Information Center for topics" on page 649

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 640
- "Updating the HTML documentation installed on your machine" on page 642
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 644
- "Searching the DB2 documentation" on page 645

Related reference:

- "Overview of DB2 Universal Database technical information" on page 627

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

Reference
DB2 reference information, such as keywords, commands, and APIs.

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- "Accessibility" on page 647
- "DB2 Information Center for topics" on page 649

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 638
- "Searching the DB2 documentation" on page 645

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

Because the following items are installed from the DB2 product CD and not the *DB2 HTML Documentation CD*, you must install the DB2 product to view these items:

- Tools help
- DB2 Quick Tour
- Release notes

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:
     ```
     e:\Program Files\sqllib\doc\htmlcd\%L\index.htm
     ```
     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:
     ```
     /cdrom/Program Files/sqllib/doc/htmlcd/%L/index.htm
     ```
     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 638
- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 644

Related reference:

- "Overview of DB2 Universal Database technical information" on page 627

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center —> Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support
3. Follow the link for version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.
   - For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:

- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 644

Related reference:

- "Overview of DB2 Universal Database technical information" on page 627

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD*, so you can install the library on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate path:

- For Windows operating systems:
  ```
  E:\Program Files\sqllib\doc\htmlcd\%L\*.*
  ```
  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:
  ```
  /cdrom:Program Files/sqllib/doc/htmlcd/%L/*.*
  ```
  where *cdrom* represents the CD-ROM drive and *%L* represents the language identifier.

Related tasks:

- "Searching the DB2 documentation" on page 645

Related reference:

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 627

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `x:Program Files\sqllib\doc\htmlcd\locale\InvalidParameterException.class` file to the `java\classes\java\security\` directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the `java\security\` subdirectory structure.
- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `/cdrom/Program Files/sqllib/doc/htmlcd/locale/InvalidParameterException.class` file to the `java/classes/java/security/` directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the `java/security/` subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:

- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser’s cache.
- Try a different version of Netscape, or a different browser.

Related tasks:

- “Searching the DB2 documentation” on page 645

Searching the DB2 documentation

To search DB2’s documentation, you need Netscape 6.1 or higher, or Microsoft’s Internet Explorer 5 or higher. Ensure that your browser’s Java support is enabled.

A pop-up search window opens when you click the search icon in the navigation toolbar of the Information Center accessed from a browser. If you are using the search for the first time it may take a minute or so to load into the search window.

Restrictions:

The following restrictions apply when you use the documentation search:

- Boolean searches are not supported. The boolean search qualifiers *and* and *or* will be ignored in a search. For example, the following searches would produce the same results:
  - servlets *and* beans
  - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will only look for the literal string *java** and would not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

Note: When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 644

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center for topics" on page 649

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 640

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation".
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 648.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 648.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 648.
- DB2 comes with documentation that is provided in an accessible format. See "Accessible Documentation" on page 648.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
Perform advanced data warehousing tasks using the Data Warehouse Center.

Development Center Tutorial for Video Online using Microsoft® Visual Basic
Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center for topics

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features:

Regularly updated documentation
Keep your topics up-to-date by downloading updated HTML.

Search
: Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree
: Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:
  - Tasks provide step-by-step instructions on how to complete a goal.
  - Concepts provide an overview of a subject.
  - Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index
: Access the information in topics and tools help from one master index. The index is organized in alphabetical order by index term.

Master glossary
: The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 638
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 640
- "Updating the HTML documentation installed on your machine" on page 642

Appendix I. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

Trademarks

Appendix J. Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Index

Part Number: CT183NA

Printed in U.S.A.

SC09-4824-00

(1P) P/N: CT183NA

Spine information:

IBM® DB2 Universal Database™

Administrative API Reference

Version 8