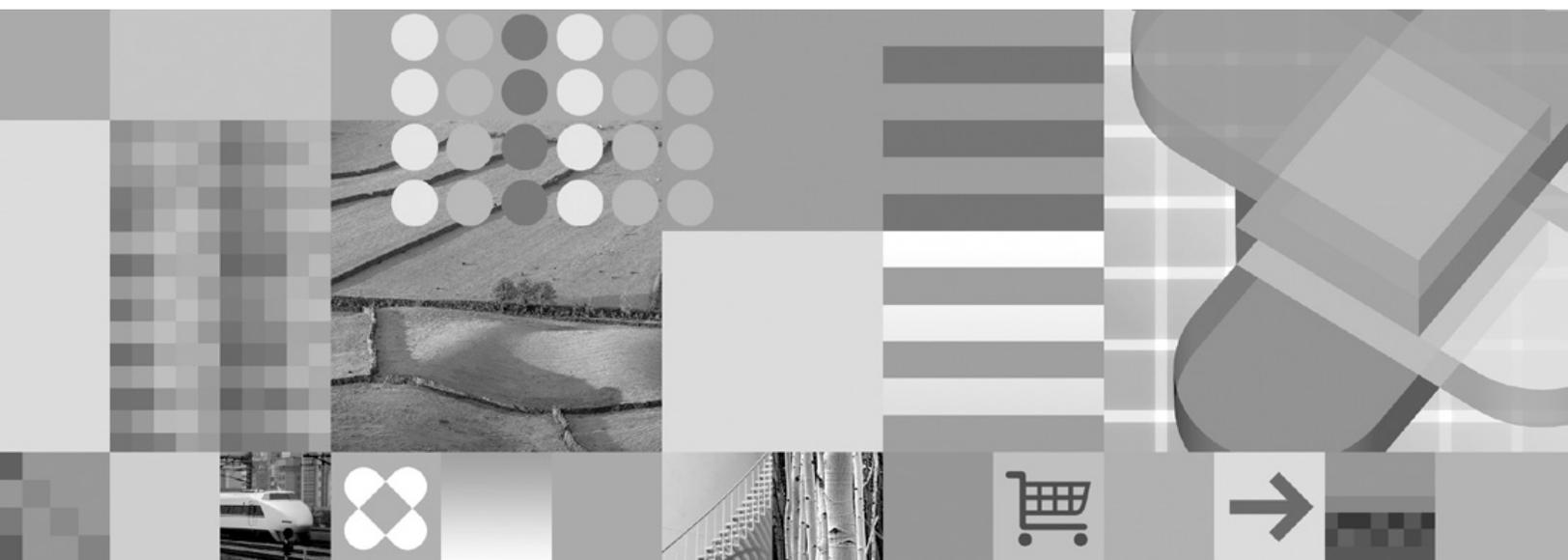




Administrative API Reference



Administrative API Reference

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

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1993, 2006. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About This Book vii

Who Should Use this Book vii

Chapter 1. DB2 APIs 1

DB2 APIs 1

Include files for DB2 API applications 19

How the API descriptions are organized. 21

db2AddContact - Add a contact to whom notification messages can be sent 24

db2AddContactGroup - Add a contact group to whom notification messages can be sent. 26

db2AddSnapshotRequest - Add a snapshot request 28

db2AdminMsgWrite - Write log messages for administration and replication function 30

db2ArchiveLog - Archive the active log file. 32

db2AutoConfig - Access the Configuration Advisor 35

db2AutoConfigFreeMemory - Free the memory allocated by the db2AutoConfig API 40

db2Backup - Back up a database or table space 41

db2CfgGet - Get the database manager or database configuration parameters 49

db2CfgSet - Set the database manager or database configuration parameters 53

db2ConvMonStream - Convert the monitor stream to the pre-version 6 format 57

db2DatabasePing - Ping the database to test network response time 60

db2DatabaseQuiesce - Quiesce the database 62

db2DatabaseRestart - Restart database 64

db2DatabaseUnquiesce - Unquiesce database 67

db2DbDirCloseScan - End a system or local database directory scan 69

db2DbDirGetNextEntry - Get the next system or local database directory entry 71

db2DbDirOpenScan - Start a system or local database directory scan 75

db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent 77

db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent 79

db2Export - Export data from a database 81

File type modifiers for the export utility 89

db2GetAlertCfg - Get the alert configuration settings for the health indicators 95

db2GetAlertCfgFree - Free the memory allocated by the db2GetAlertCfg API. 100

db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent. 101

db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent 103

db2GetContacts - Get the list of contacts to whom notification messages can be sent. 105

db2GetHealthNotificationList - Get the list of contacts to whom health alert notifications can be sent 107

db2GetRecommendations - Get recommendations to resolve a health indicator in alert state 109

db2GetRecommendationsFree - Free the memory allocated by the db2GetRecommendations API 112

db2GetSnapshot - Get a snapshot of the database manager operational status 113

db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API 117

db2GetSyncSession - Get a satellite synchronization session identifier 120

db2HADRStart - Start high availability disaster recovery (HADR) operations 121

db2HADRStop - Stop high availability disaster recovery (HADR) operations 123

db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database 125

db2HistoryCloseScan - End the history file scan 127

db2HistoryGetEntry - Get the next entry in the history file 129

db2HistoryOpenScan - Start a history file scan 132

db2HistoryUpdate - Update a history file entry 136

db2Import - Import data into a table, hierarchy, nickname or view 139

File type modifiers for the import utility 153

db2Inspect - Inspect database for architectural integrity 164

db2InstanceQuiesce - Quiesce instance 170

db2InstanceStart - Start instance 173

db2InstanceStop - Stop instance 179

db2InstanceUnquiesce - Unquiesce instance 183

db2LdapCatalogDatabase - Register the database on the LDAP server 185

db2LdapCatalogNode - Provide an alias for node name in LDAP server 188

db2LdapDeregister - Deregister the DB2 server and cataloged databases from the LDAP server 190

db2LdapRegister - Register the DB2 server on the LDAP server 192

db2LdapUncatalogDatabase - Deregister database from LDAP server. 196

db2LdapUncatalogNode - Delete alias for node name from LDAP server. 198

db2LdapUpdate - Update the attributes of the DB2 server on the LDAP server 200

db2LdapUpdateAlternateServerForDB - Update the alternate server for the database on the LDAP server 203

db2Load - Load data into a table 205

File type modifiers for the load utility 225

Delimiter restrictions for moving data 237

db2LoadQuery - Get the status of a load operation 239

db2MonitorSwitches - Get or update the monitor switch settings	246	sqlbftcq - Fetch the query data for rows in a table space container	363
db2Prune - Delete the history file entries or log files from the active log path	249	sqlbftpq - Fetch the query data for rows in a table space	365
db2QuerySatelliteProgress - Get the status of a satellite synchronization session	252	sqlbgtss - Get table space usage statistics	367
db2ReadLog - Extracts log records	254	sqlbmtsq - Get the query data for all table spaces	369
db2ReadLogNoConn - Read the database logs without a database connection.	259	sqlbotcq - Open a table space container query	371
db2ReadLogNoConnInit - Initialize reading the database logs without a database connection	262	sqlbotsq - Open a table space query	373
db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection	264	sqlbstpq - Get information about a single table space	375
db2Recover - Restore and roll forward a database	265	sqlbstsc - Set table space containers	377
db2Reorg - Reorganize an index or a table.	271	sqlbtcq - Get the query data for all table space containers	379
db2ResetAlertCfg - Reset the alert configuration of health indicators	279	sqlcspqy - List DRDA indoubt transactions	381
db2ResetMonitor - Reset the database system monitor data	281	sqle_activate_db - Activate database.	383
db2Restore - Restore a database or table space	284	sqle_deactivate_db - Deactivate database	386
db2Rollforward - Roll forward a database	296	sqleaddn - Add a database partition server to the partitioned database environment	388
db2Runstats - Update statistics for tables and indexes	306	sqleatcp - Attach to instance and change password	391
db2SelectDB2Copy - Select the DB2 copy to be used by your application	316	sqleatin - Attach to instance	394
db2SetSyncSession - Set satellite synchronization session	318	sqleAttachToCtx - Attach to context	397
db2SetWriteForDB - Suspend or resume I/O writes for database	320	sqleBeginCtx - Create and attach to an application context	398
db2SpmListIndTrans - List SPM indoubt transactions	322	sqlecadb - Catalog a database in the system database directory.	400
db2SyncSatellite - Start satellite synchronization	325	sqlecra - Create a database on a database partition server	406
db2SyncSatelliteStop - Pause satellite synchronization	326	sqlecrea - Create database	408
db2SyncSatelliteTest - Test whether a satellite can be synchronized	327	sqlectnd - Catalog an entry in the node directory	415
db2UpdateAlertCfg - Update the alert configuration settings for health indicators	328	sqledcgd - Change a database comment in the system or local database directory	419
db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory.	334	sqleDetachFromCtx - Detach from context.	422
db2UpdateContact - Update the attributes of a contact	336	sqledpan - Drop a database on a database partition server	423
db2UpdateContactGroup - Update the attributes of a contact group.	338	sqledrpd - Drop database	425
db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent	340	sqledrpn - Check whether a database partition server can be dropped	427
db2UtilityControl - Set the priority level of running utilities	342	sqledtin - Detach from instance	429
db2XaGetInfo - Get information for a resource manager	344	sqleEndCtx - Detach from and free the memory associated with an application context	430
db2XaListIndTrans - List indoubt transactions	345	sqlefmem - Free the memory allocated by the sqlbftcq and sqlbmtsq API	432
sqlabndx - Bind application program to create a package	350	sqlefrce - Force users and applications off the system	433
sqlaintp - Get error message	353	sqlegdad - Catalog a database in the database connection services (DCS) directory	436
sqlaprep - Precompile application program	355	sqlegdcl - End a database connection services (DCS) directory scan	438
sqlarbnd - Rebind package	358	sqlegdel - Uncatalog a database from the database connection services (DCS) directory	439
sqlbctcq - Close a table space container query	361	sqlegdge - Get a specific entry in the database connection services (DCS) directory	441
sqlbctsq - Close a table space query	362	sqlegdgt - Get database connection services (DCS) directory entries	443
		sqlegdsc - Start a database connection services (DCS) directory scan	445
		sqleGetCurrentCtx - Get current context	447
		sqlegins - Get current instance.	448
		sqleInterruptCtx - Interrupt context	450
		sqleintr - Interrupt application requests.	451
		sqleisig - Install signal handler	453

sqlmngdb - Migrate previous version of DB2 database to current version	455
sqlencls - End a node directory scan	457
sqlengne - Get the next node directory entry	458
sqlenops - Start a node directory scan	460
sqlqryc - Query client connection settings	462
sqlqryi - Query client information	464
sqlsact - Set accounting string	466
sqlsdeg - Set the maximum runtime intra-partition parallelism level or degree for SQL statements	468
sqlsetc - Set client connection settings	470
sqlseti - Set client information	473
sqlSetTypeCtx - Set application context type	476
sqluncd - Uncatalog a database from the system database directory	478
sqluncn - Uncatalog an entry from the node directory	480
sqlgaddr - Get the address of a variable	482
sqlgdref - Dereference an address	483
sqlgmcpy - Copy data from one memory area to another	484
sqlogstt - Get the SQLSTATE message	485
sqluadai - Get current user's authorities	487
sqludrtd - Redistribute data across a database partition group	489
sqlugrpn - Get the database partition server number for a row	493
sqlugtpi - Get table distribution information	496
sqluvqdp - Quiesce table spaces for a table	498
sqlxhfrg - Forget transaction status	501
sqlxphcm - Commit an indoubt transaction	502
sqlxphrl - Roll back an indoubt transaction	504

Chapter 2. Data structures used by multiple APIs 507

db2HistoryData	507
sql_authorizations	513
sql_dir_entry	515
SQLB_TBS_STATS	517
SQLB_TBSCONTQRY_DATA	519
SQLB_TBSPQRY_DATA	521
SQLCA	526
sqlchar	527
SQLDA	528
sqldcol	530
sql_e_addn_options	533
sql_e_client_info	535
sql_e_conn_setting	538
sql_e_node_cplic	541
sql_e_node_ipxspix	542
sql_e_node_local	543
sql_e_node_netb	544
sql_e_node_npip	545
sql_e_node_struct	546
sql_e_node_tcpip	548
sql_ebdbdesc	549
sql_ebdbdescext	556
sql_ebdbterritoryinfo	563
sql_einfo	564
sql_eupd	567
sql_lob	574

sqlma	575
sqlopt	579
SQLU_LSN	581
sqlu_media_list	582
SQLU_RLOG_INFO	587
sqlupi	588
SQLXA_XID	590

Chapter 3. DB2 database system plug-ins for customizing database management. 591

DB2 database system plug-ins for customizing database management	591
Security plug-ins	592
Security plug-ins	592
Developing security plug-ins	614
Security plug-in APIs	625
Backup and restore plug-ins for vendor storage devices	671
DB2 APIs for backup and restore to storage managers	671
sqluvint - Initialize and link to a vendor device	680
sqluvget - Read data from a vendor device	685
sqluvput - Write data to a vendor device	687
sqluvend - Unlink a vendor device and release its resources	689
sqluvdel - Delete committed session	691
db2VendorQueryApiVersion - Get the supported level of the vendor storage API	693
db2VendorGetNextObj - Get next object on device	694
DB2_info	696
Vendor_info	699
Init_input	701
Init_output	703
Data	704
Return_code	705
Compression plug-ins for backup and restore	705
DB2 APIs for using compression with backup and restore operations	705
InitCompression - Initialize the compression library	708
GetSavedBlock - Get the vendor of block data for the backup image	709
Compress - Compress a block of data	710
GetMaxCompressedSize - Estimate largest possible buffer size	712
TermCompression - Stop the compression library	713
InitDecompression - Initialize the decompression library	714
Decompress - Decompress a block of data	715
TermDecompression - Stop the decompression library	716
COMPR_DB2INFO	717
COMPR_PIINFO	719
COMPR_CB	721

Appendix A. Precompiler customization APIs 723

Appendix B. DB2 log records 725

Log Manager Header. 727
Data Manager Log Records. 729
 Initialize Table 730
 Import Replace (Truncate) 732
 Activate Not Logged Initially 732
 Rollback Insert 733
 Reorg Table 733
 Create index, drop index log records 733
 Create table, drop table, rollback create table,
 rollback drop table log records 734
 Alter Table Attribute 734
 Alter Table Add Columns, Rollback Add
 Columns 735
 Alter Column Attribute 736
 Undo Alter Column Attribute 736
 Insert Record, Rollback Delete Record, Rollback
 Update Record 736
 Formatted User Data record for a table without
 VALUE COMPRESSION. 737
 Formatted User Data record for table with
 VALUE COMPRESSION. 739
 Insert Record to Empty Page, Delete Record to
 Empty Page, Rollback Delete Record to Empty
 Page, Rollback Insert Record to Empty Page . . . 739
 Update Record 740
 Rename of a Table or Schema Log Record . . . 741

 Undo Rename of a Table or Schema Log Record 741
Long Field Manager Log Records. 741
 Add/Delete/Non-update Long Field Record 742
Transaction Manager Log Records 743
 Normal commit 743
 Heuristic Commit 744
 MPP Coordinator Commit 744
 MPP Subordinator Commit. 744
 Normal Abort 745
 Heuristic Abort. 745
 Local Pending List. 746
 Global Pending List 746
 XA Prepare 746
 MPP Subordinator Prepare 747
 TM Prepare 748
 Backout free. 748
Utility Manager Log Records 749
 Application migration 751
 REXX APIs 754
 DB2 Database technical information 756
 Notices 765

Index 769

Contacting IBM 779

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. DB2 APIs

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 12.

COBOL Sample Programs with DB2 APIs

Table 3 on page 15.

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

API Type	DB2 API	Include File	Sample Programs
Database control APIs	db2DatabaseQuiesce API - Quiesce the database	db2ApiDf	n/a
Database control APIs	db2DatabaseUnquiesce API - Unquiesce database	db2ApiDf	n/a
Database control APIs	db2DatabaseRestart API - Restart database	db2ApiDf	C: dbconn.sqc C++: dbconn.sqC
Database control APIs	sqlcrea API - Create database	sqlenv	C: dbcreate.c dbrecov.sqc dbsample.sqc C++: dbcreate.C dbrecov.sqC COBOL: db_udcs.cb1 dbconf.cb1 ebcdicdb.cb1
Database control APIs	sqlcran API - Create a database on a database partition server	sqlenv	n/a
Database control APIs	sqledrpd API - Drop database	sqlenv	C: dbcreate.c C++: dbcreate.C COBOL: dbconf.cb1
Database control APIs	sqledpan API - Drop a database on a database partition server	sqlenv	n/a
Database control APIs	sqlmgdb API - Migrate previous version of DB2 database to current version	sqlenv	C: dbmigrat.c C++: dbmigrat.C COBOL: migrate.cb1
Database control APIs	db2XaListIndTrans API - List indoubt transactions	db2ApiDf	n/a
Database control APIs	sql_activate_db API - Activate database	sqlenv	n/a
Database control APIs	sql_deactivate_db API - Deactivate database	sqlenv	n/a

DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
Database control APIs	sqlcspqy API - List DRDA indoubt transactions	sqlxa	n/a
Database control APIs	db2SetWriteForDB API - Suspend or resume I/O writes for database	db2ApiDf	n/a
Database control APIs	sqlfrce API - Force users and applications off the system	sqlenv	C: dbconn.sqc dbsample.sqc instart.c C++: dbconn.sqC instart.C COBOL: dbstop.cb1
Instance control APIs	db2InstanceStart API - Start instance	db2ApiDf	C: instart.c C++: instart.C
Instance control APIs	db2InstanceStop API - Stop instance	db2ApiDf	C: instart.c C++: instart.C
Instance control APIs	db2InstanceQuiesce API - Quiesce instance	db2ApiDf	n/a
Instance control APIs	db2InstanceUnquiesce API - Unquiesce instance	db2ApiDf	n/a
Instance control APIs	sqlleatin API - Attach to instance	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cb1
Instance control APIs	sqlleatcp API - Attach to instance and change password	sqlenv	C: inattach.c C++: inattach.C COBOL: dbinst.cb1
Instance control APIs	sqledtin API - Detach from instance	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cb1
Instance control APIs	sqllegins API - Get current instance	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbinst.cb1
Instance control APIs	sqluadai API - Get current user's authorities	sqlutil	n/a
Instance control APIs	db2UtilityControl API - Set the priority level of running utilities	db2ApiDf	n/a
Database manager and database configuration APIs	db2CfgGet API - Get the database manager or database 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
Database manager and database configuration APIs	db2CfgSet API - Set the database manager or database configuration parameters	db2ApiDf	C: dbinfo.c dbrecov.sqc ininfo.c C++: dbinfo.C dbrecov.sqC ininfo.C
Database manager and database configuration APIs	db2AutoConfig API - Access the Configuration Advisor	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC
Database manager and database configuration APIs	db2AutoConfigFreeMemory API - Free the memory allocated by the db2AutoConfig API	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC
Database monitoring APIs	db2GetSnapshotSize API - Estimate the output buffer size required for the db2GetSnapshot API	db2ApiDf	n/a
Database monitoring APIs	db2AddSnapshotRequest API - Add a snapshot request	db2ApiDf	n/a
Database monitoring APIs	db2MonitorSwitches API - Get or update the monitor switch settings	db2ApiDf	C: utilsnap.c C++: utilsnap.C

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

API Type	DB2 API	Include File	Sample Programs
Database monitoring APIs	db2GetSnapshot API - Get a snapshot of the database manager operational status	db2ApiDf	C: utilsnap.c C++: utilsnap.C
Database monitoring APIs	db2ResetMonitor API - Reset the database system monitor data	db2ApiDf	n/a
Database monitoring APIs	db2ConvMonStream API - Convert the monitor stream to the pre-version 6 format	db2ApiDf	n/a
Database monitoring APIs	db2Inspect API - Inspect database for architectural integrity	db2ApiDf	n/a
Database health monitoring APIs	db2AddContact API - Add a contact to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2AddContactGroup API - Add a contact group to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2DropContact API - Remove a contact from the list of contacts to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2DropContactGroup API - Remove a contact group from the list of contacts to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2GetAlertCfg API - Get the alert configuration settings for the health indicators	db2ApiDf	n/a
Database health monitoring APIs	db2GetAlertCfgFree API - Free the memory allocated by the db2GetAlertCfg API	db2ApiDf	n/a
Database health monitoring APIs	db2GetContactGroup API - Get the list of contacts in a single contact group to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2GetContactGroups API - Get the list of contact groups to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2GetContacts API - Get the list of contacts to whom notification messages can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2GetHealthNotificationList API - Get the list of contacts to whom health alert notifications can be sent	db2ApiDf	n/a
Database health monitoring APIs	db2ResetAlertCfg API - Reset the alert configuration of health indicators	db2ApiDf	n/a
Database health monitoring APIs	db2UpdateAlertCfg API - Update the alert configuration settings for health indicators	db2ApiDf	n/a
Database health monitoring APIs	db2UpdateContact API - Update the attributes of a contact	db2ApiDf	n/a
Database health monitoring APIs	db2UpdateContactGroup API - Update the attributes of a contact group	db2ApiDf	n/a
Database health monitoring APIs	db2UpdateHealthNotificationList API - Update the list of contacts to whom health alert notifications can be sent	db2ApiDf	n/a

DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
Database health monitoring APIs	db2GetSnapshot API - Get a snapshot of the database manager operational status	db2ApiDf	C: utilsnap.c C++: utilsnap.C
Database health monitoring APIs	db2GetSnapshotSize API - Estimate the output buffer size required for the db2GetSnapshot API	db2ApiDf	n/a
Database health monitoring APIs	db2GetRecommendations API - Get recommendations to resolve a health indicator in alert state	db2ApiDf	n/a
Database health monitoring APIs	db2GetRecommendationsFree API - Free the memory allocated by the db2GetRecommendations API	db2ApiDf	n/a
Data movement APIs	db2Export API - Export data from a database	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: expsamp.sqb impexp.sqb tload.sqb
Data movement APIs	db2Import API - Import data into a table, hierarchy, nickname or view	db2ApiDf	C: dtformat.sqc tbmove.sqc C++: tbmove.sqC COBOL: expsamp.sqb impexp.sqb
Data movement APIs	db2Load API - Load data into a table	db2ApiDf	C: dtformat.sqc tload.sqc tbmove.sqc C++: tbmove.sqC
Data movement APIs	db2LoadQuery API - Get the status of a load operation	db2ApiDf	C: tbmove.sqc C++: tbmove.sqC COBOL: loadqry.sqb
Data movement APIs	sqludrdt API - Redistribute data across a database partition group	sqlutil	n/a
Recovery APIs	db2Backup API - Back up a database or table space	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2Restore API - Restore a database or table space	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2Recover API - Restore and roll forward a database	db2ApiDf	n/a
Recovery APIs	db2Rollforward API - Roll forward a database	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2HistoryOpenScan API - Start a history file scan	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2HistoryGetEntry API - Get the next entry in the history file	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2HistoryCloseScan API - End the history file scan	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2Prune API - Delete the history file entries or log files from the active log path	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2HistoryUpdate API - Update a history file entry	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Recovery APIs	db2ArchiveLog API - Archive the active log file	db2ApiDf	n/a
High Availability Disaster Recovery (HADR) APIs	db2HADRStart API - Start high availability disaster recovery (HADR) operations	db2ApiDf	n/a
High Availability Disaster Recovery (HADR) APIs	db2HADRStop API - Stop high availability disaster recovery (HADR) operations	db2ApiDf	n/a

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

API Type	DB2 API	Include File	Sample Programs
High Availability Disaster Recovery (HADR) APIs	db2HADRTakeover API - Instruct a database to take over as the high availability disaster recovery (HADR) primary database	db2ApiDf	n/a
Database directory and DCS directory management APIs	sqlcadb API - Catalog a database in the system database directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcatt.cb1
Database directory and DCS directory management APIs	sqluncd API - Uncatalog a database from the system database directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcatt.cb1
Database directory and DCS directory management APIs	sqlgdad API - Catalog a database in the database connection services (DCS) directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscatt.cb1
Database directory and DCS directory management APIs	sqlgdel API - Uncatalog a database from the database connection services (DCS) directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscatt.cb1
Database directory and DCS directory management APIs	sqldcdg API - Change a database comment in the system or local database directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcmt.cb1
Database directory and DCS directory management APIs	db2DbDirOpenScan API - Start a system or local database directory scan	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcatt.cb1 dbcmt.cb1
Database directory and DCS directory management APIs	db2DbDirGetNextEntry API - Get the next system or local database directory entry	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcatt.cb1 dbcmt.cb1
Database directory and DCS directory management APIs	db2DbDirCloseScan API - End a system or local database directory scan	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcatt.cb1 dbcmt.cb1
Database directory and DCS directory management APIs	sqlgdsc API - Start a database connection services (DCS) directory scan	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscatt.cb1
Database directory and DCS directory management APIs	sqlgdgt API - Get database connection services (DCS) directory entries	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscatt.cb1
Database directory and DCS directory management APIs	sqlgdcl API - End a database connection services (DCS) directory scan	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscatt.cb1

DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
Database directory and DCS directory management APIs	sqlqedge API - Get a specific entry in the database connection services (DCS) directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
Database directory and DCS directory management APIs	db2UpdateAlternateServerForDB API - Update the alternate server for a database alias in the system database directory	db2ApiDf	n/a
Client/server management APIs	sqlqryc API - Query client connection settings	sqlenv	C: cli_info.c C++: cli_info.C COBOL: client.cbl
Client/server management APIs	sqlqryi API - Query client information	sqlenv	C: cli_info.c C++: cli_info.C
Client/server management APIs	sqlsetc API - Set client connection settings	sqlenv	C: cli_info.c dbcfg.sqc dbmcon.sqc C++: cli_info.C dbcfg.sqC dbmcon.sqC COBOL: client.cbl
Client/server management APIs	sqlseti API - Set client information	sqlenv	C: cli_info.c C++: cli_info.C
Client/server management APIs	sqlsact API - Set accounting string	sqlenv	COBOL: setact.cbl
Client/server management APIs	db2DatabasePing API - Ping the database to test network response time	db2ApiDf	n/a
Client/server management APIs	sqlsig API - Install signal handler	sqlenv	COBOL: dbcmt.cbl
Client/server management APIs	sqlintr API - Interrupt application requests	sqlenv	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapRegister API - Register the DB2 server on the LDAP server	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapUpdate API - Update the attributes of the DB2 server on the LDAP server	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapDeregister API - Deregister the DB2 server and cataloged databases from the LDAP server	db2ApiDf	n/a

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

API Type	DB2 API	Include File	Sample Programs
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapCatalogNode API - Provide an alias for node name in LDAP server	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapUncatalogNode API - Delete alias for node name from LDAP server	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapCatalogDatabase API - Register the database on the LDAP server	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapUncatalogDatabase API - Deregister database from LDAP server	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	db2LdapUpdateAlternateServerForDB API - Update the alternate server for the database on the LDAP server	db2ApiDf	n/a
Application programming and preparation APIs	sqlaintp API - Get error message	sql	C: dbcfg.sqc utilapi.c C++: dbcfg.sqc utilapi.C COBOL: checkerr.cb1
Application programming and preparation APIs	sqlogstt API - Get the SQLSTATE message	sql	C: utilapi.c C++: utilapi.C COBOL: checkerr.cb1
Application programming and preparation APIs	sqlleisig API - Install signal handler	sqlenv	COBOL: dbcmt.cb1
Application programming and preparation APIs	sqlintr API - Interrupt application requests	sqlenv	n/a
Application programming and preparation APIs	sqlaprep API - Precompile application program	sql	C: dbpkg.sqc C++: dbpkg.sqc
Application programming and preparation APIs	sqlabndx API - Bind application program to create a package	sql	C: dbpkg.sqc dbsample.sqc C++: dbpkg.sqc

DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
Application programming and preparation APIs	sqlarbind API - Rebind package	sql	C: dbpkg.sqc C++: dbpkg.sqc COBOL: rebind.sqb
COBOL, FORTRAN and REXX application specific APIs	sqlgaddr API - Get the address of a variable	sqlutil	n/a
COBOL, FORTRAN and REXX application specific APIs	sqlgdref API - Dereference an address	sqlutil	n/a
COBOL, FORTRAN and REXX application specific APIs	sqlgncpy API - Copy data from one memory area to another	sqlutil	n/a
Table space and table management APIs	sqlbtcq API - Get the query data for all table space containers	sqlutil	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqc tsinfo.sqc COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	sqlbotcq API - Open a table space container query	sqlutil	C: tsinfo.sqc C++: tsinfo.sqc COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	sqlbftcq API - Fetch the query data for rows in a table space container	sqlutil	C: tsinfo.sqc C++: tsinfo.sqc COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	sqlbctcq API - Close a table space container query	sqlutil	C: tsinfo.sqc C++: tsinfo.sqc COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	sqlbstsc API - Set table space containers	sqlutil	C: dbrecov.sqc C++: dbrecov.sqc COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	sqlbmtsq API - Get the query data for all table spaces	sqlutil	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqc tsinfo.sqc COBOL: tabspace.sqb tspace.sqb
Table space and table management APIs	sqlbstpq API - Get information about a single table space	sqlutil	C: tsinfo.sqc C++: tsinfo.sqc COBOL: tabspace.sqb tspace.sqb
Table space and table management APIs	sqlbotsq API - Open a table space query	sqlutil	C: tsinfo.sqc C++: tsinfo.sqc COBOL: tabspace.sqb tspace.sqb
Table space and table management APIs	sqlbftpq API - Fetch the query data for rows in a table space	sqlutil	C: tsinfo.sqc C++: tsinfo.sqc COBOL: tabspace.sqb tspace.sqb
Table space and table management APIs	sqlbctsq API - Close a 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)

API Type	DB2 API	Include File	Sample Programs
Table space and table management APIs	sqlbgtss API - Get table space usage statistics	sqlutil	C: tsinfo.sqc C++: tsinfo.sqC COBOL: tabspace.sqb tspace.sqb
Table space and table management APIs	sqluvqdp API - Quiesce table spaces for a table	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: tload.sqb
Table space and table management APIs	db2Runstats API - Update statistics for tables and indexes	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
Table space and table management APIs	db2Reorg API - Reorganize an index or a table	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
Table space and table management APIs	sqlfmem API - Free the memory allocated by the sqlbtcq and sqlbmtsq API	sqlenv	C: dbrecov.sqc tsinfo.sqc C++: dbrecov.sqC tsinfo.sqC COBOL: tabscont.sqb tabspace.sqb tspace.sqb
Node directory management APIs	sqlctnd API - Catalog an entry in the node directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cb1
Node directory management APIs	sqluncn API - Uncatalog an entry from the node directory	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cb1
Node directory management APIs	sqlenops API - Start a node directory scan	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cb1
Node directory management APIs	sqlengne API - Get the next node directory entry	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cb1
Node directory management APIs	sqlencls API - End a node directory scan	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cb1
Node directory management APIs	db2UpdateAlternateServerForDB API - Update the alternate server for a database alias in the system database directory	db2ApiDf	n/a
Satellite synchronization APIs	db2GetSyncSession API - Get a satellite synchronization session identifier	db2ApiDf	n/a
Satellite synchronization APIs	db2QuerySatelliteProgress API - Get the status of a satellite synchronization session	db2ApiDf	n/a
Satellite synchronization APIs	db2SetSyncSession API - Set satellite synchronization session	db2ApiDf	n/a
Satellite synchronization APIs	db2SyncSatellite API - Start satellite synchronization	db2ApiDf	n/a
Satellite synchronization APIs	db2SyncSatelliteStop API - Pause satellite synchronization	db2ApiDf	n/a
Satellite synchronization APIs	db2SyncSatelliteTest API - Test whether a satellite can be synchronized	db2ApiDf	n/a

DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
Read log files APIs	db2ReadLog API - Extracts log records	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
Read log files APIs	db2ReadLogNoConn API - Read the database logs without a database connection	db2ApiDf	n/a
Read log files APIs	db2ReadLogNoConnInit API - Initialize reading the database logs without a database connection	db2ApiDf	n/a
Read log files APIs	db2ReadLogNoConnTerm API - Terminate reading the database logs without a database connection	db2ApiDf	n/a
Indoubt transaction management APIs	db2XaListFndTrans API - List indoubt transactions	db2ApiDf	n/a
Indoubt transaction management APIs	sqlxhfrg API - Forget transaction status	sqlxa	n/a
Indoubt transaction management APIs	sqlxphcm API - Commit an indoubt transaction	sqlxa	n/a
Indoubt transaction management APIs	sqlxphrl API - Roll back an indoubt transaction	sqlxa	n/a
Indoubt transaction management APIs	sqlcspqy API - List DRDA indoubt transactions	sqlxa	n/a
APIs for obtaining concurrent access to a database	sqlAttachToCtx API - Attach to context	sql	C: dbthrds.sqc C++: dbthrds.sqC
APIs for obtaining concurrent access to a database	sqlBeginCtx API - Create and attach to an application context	sql	C: dbthrds.sqc C++: dbthrds.sqC
APIs for obtaining concurrent access to a database	sqlDetachFromCtx API - Detach from context	sql	C: dbthrds.sqc C++: dbthrds.sqC
APIs for obtaining concurrent access to a database	sqlEndCtx API - Detach from and free the memory associated with an application context	sql	n/a
APIs for obtaining concurrent access to a database	sqlGetCurrentCtx API - Get current context	sql	n/a

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

API Type	DB2 API	Include File	Sample Programs
APIs for obtaining concurrent access to a database	sqlInterruptCtx API - Interrupt context	sql	n/a
APIs for obtaining concurrent access to a database	sqlSetTypeCtx API - Set application context type	sql	C: dbthrds.sqc C++: dbthrds.sqC
Database partition management APIs	sqladdn API - Add a database partition server to the partitioned database environment	sqlenv	n/a
Database partition management APIs	sqledrpn API - Check whether a database partition server can be dropped	sqlenv	n/a
Database partition management APIs	sqlcran API - Create a database on a database partition server	sqlenv	n/a
Database partition management APIs	sqledpan API - Drop a database on a database partition server	sqlenv	n/a
Database partition management APIs	sqlsdeg API - Set the maximum runtime intra-partition parallelism level or degree for SQL statements	sqlenv	C: ininfo.c C++: ininfo.C
Database partition management APIs	sqlugtpi API - Get table distribution information	sqlutil	n/a
Database partition management APIs	sqlugrpn API - Get the database partition server number for a row	sqlutil	n/a
Database partition management APIs	sqludrtd API - Redistribute data across a database partition group	sqlutil	n/a
Miscellaneous APIs	db2AdminMsgWrite API - Write log messages for administration and replication function	db2ApiDf	n/a
Miscellaneous APIs	db2XaGetInfo API - Get information for a resource manager	sqlxa	n/a

DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
<p>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:</p>			
<p>C/C++ (UNIX): sqllib/include</p>			
<p>C/C++ (Windows): sqllib\include</p>			
<p>COBOL (UNIX): sqllib/include/cobol_a sqllib/include/cobol_i sqllib/include/cobol_mf</p>			
<p>COBOL (Windows): sqllib\include\cobol_a sqllib\include\cobol_i sqllib\include\cobol_mf</p>			

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

Sample Program	Included APIs
cli_info.c, cli_info.C	<ul style="list-style-type: none"> • sqlesetc API - Set client connection settings • sqleseti API - Set client information • sqleqryc API - Query client connection settings • sqleqryi API - Query client information
dbauth.sqc, dbauth.sqC	<ul style="list-style-type: none"> • sqluadau API - Get current user's authorities
dbcfg.sqc, dbcfg.sqC	<ul style="list-style-type: none"> • db2AutoConfig API - Access the Configuration Advisor • db2AutoConfigFreeMemory API - Free the memory allocated by the db2AutoConfig API • sqlesetc API - Set client connection settings • sqlaintp API - Get error message
dbconn.sqc, dbconn.sqC	<ul style="list-style-type: none"> • db2DatabaseRestart API - Restart database • sqlfrcce API - Force users and applications off the system
dbcreate.c, dbcreate.C	<ul style="list-style-type: none"> • sqlcrea API - Create database • sqldrpd API - Drop database
dbinfo.c, dbinfo.C	<ul style="list-style-type: none"> • db2CfgGet API - Get the database manager or database configuration parameters • db2CfgSet API - Set the database manager or database configuration parameters
dbmcon.sqc, dbmcon.sqC	<ul style="list-style-type: none"> • sqlesetc API - Set client connection settings
dbmigrat.c, dbmigrat.C	<ul style="list-style-type: none"> • sqlmgdb API - Migrate previous version of DB2 database to current version
dbpkg.sqc, dbpkg.sqC	<ul style="list-style-type: none"> • sqlaprep API - Precompile application program • sqlabndx API - Bind application program to create a package • sqlarbnd API - Rebind package

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

Sample Program	Included APIs
dbrecov.sqc, dbrecov.sqC	<ul style="list-style-type: none"> • db2HistoryCloseScan API - End the history file scan • db2HistoryGetEntry API - Get the next entry in the history file • db2HistoryOpenScan API - Start a history file scan • db2HistoryUpdate API - Update the history file entry • db2Prune API - Delete the history file entries or log files from the active log path • db2CfgGet API - Get the database manager or database configuration parameters • db2CfgSet API - Set the database manager or database configuration parameters • sqlbmtsq API - Get the query data for all table spaces • sqlbstsc API - Set table space containers • sqlbtcq API - Get the query data for all table space containers • sqlcrea API - Create database • sqledrpd API - Drop database • sqlfmem API - Free the memory allocated by the sqlbtcq and sqlbmtsq APIs • db2Backup API - Back up a database or table space • db2Restore API - Restore a database or table space • db2ReadLog API - Asynchronous read log • db2ReadLogNoConn API - Read log without a database connection • db2Rollforward API - Roll forward a database
dbsample.sqc	<ul style="list-style-type: none"> • db2DatabaseRestart API - Restart database • sqlcrea API - Create database • sqlfrcce API - Force users and applications off the system • sqlabndx API - Bind application program to create a package
dbthrs.sqc, dbthrs.sqC	<ul style="list-style-type: none"> • sqlAttachToCtx API - Attach to context • sqlBeginCtx API - Create and attach to an application context • sqlDetachFromCtx API - Detach from context • sqlSetTypeCtx API - Set application context type
dtformat.sqc	<ul style="list-style-type: none"> • db2Load API - Load data into a table • db2Import API - Import data into a table, hierarchy, nickname or view
inattach.c, inattach.C	<ul style="list-style-type: none"> • sqlatcp API - Attach to instance and change password • sqlatin API - Attach to instance • sqledtin API - Detach from instance
inauth.sqc, inauth.sqC	<ul style="list-style-type: none"> • db2CfgGet API - Get the database manager or database configuration parameters • sqluadcu API - Get current user's authorities

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

Sample Program	Included APIs
ininfo.c, ininfo.C	<ul style="list-style-type: none"> • db2CfgGet API - Get the database manager or database configuration parameters • db2CfgSet API - Set the database manager or database configuration parameters • sqlgins API - Get current instance • sqlctnd API - Catalog an entry in the node directory • sqlenops API - Start a node directory scan • sqlengne API - Get the next node directory entry • sqlencls API - End a node directory scan • sqlleuncn API - Uncatalog an entry from the node directory • sqlacadb API - Catalog a database in the system database directory • db2DbDirOpenScan API - Start a system or local database directory scan • db2DbDirGetNextEntry API - Get the next system or local database directory entry • sqlעדcgd API - Change a database comment in the system or local database directory • db2DbDirCloseScan API - End a system or local database directory scan • sqlleuncd API - Uncatalog a database from the system database directory • sqlעדgdad API - Catalog a database in the database connection services (DCS) directory • sqlעדgdsc API - Start a database connection services (DCS) directory scan • sqlעדgde API - Get a specific entry in the database connection services (DCS) directory • sqlעדgdgt API - Get database connection services (DCS) directory entries • sqlעדgdcl API - End a database connection services (DCS) directory scan • sqlעדgdel API - Uncatalog a database from the database connection services (DCS) directory • sqlעדsdeg API - Set the maximum runtime intra-partition parallelism level or degree for SQL statements
instart.c, instart.C	<ul style="list-style-type: none"> • sqlעfrce API - Force users and applications off the system • db2InstanceStart API - Start instance • db2InstanceStop API - Stop instance
tbmove.sqc, tbmove.sqC	<ul style="list-style-type: none"> • db2Export API - Export data from a database • db2Import API - Import data into a table, hierarchy, nickname or view • sqlעuvqdp API - Quiesce table spaces for a table • db2Load API - Load data into a table • db2LoadQuery API - Get the status of a load operation
tbreorg.sqc, tbreorg.sqC	<ul style="list-style-type: none"> • db2Reorg API - Reorganize an index or a table • db2Runstats API - Update statistics about the characteristics of a table and associated indexes

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

Sample Program	Included APIs
tscreate.sqc, tscreate.sqC	<ul style="list-style-type: none"> db2CfgGet API - Get the database manager or database configuration parameters
tsinfo.sqc, tsinfo.sqC	<ul style="list-style-type: none"> sqlbstpq API - Get information about a single table space sqlbgtss API - Get table space usage statistics sqlbmtsq API - Get the query data for all table spaces sqlfmem API - Free the memory allocated by the sqlbtcq and sqlbmtsq APIs sqlbotsq API - Open a table space query sqlbftpq API - Fetch the query data for rows in a table space sqlbctsq API - Close a table space query sqlbtcq API - Get the query data for all table space containers sqlbotcq API - Open a table space container query sqlbftcq API - Fetch the query data for rows in a table space container sqlbctcq API - Close a table space container query
utilapi.c, utilapi.C	<ul style="list-style-type: none"> sqlaintp API - Get error message sqlogstt API - Get the SQLSTATE message sqleatin API - Attach to instance sqledtin API - Detach from instance
utisnap.c, utisnap.C	<ul style="list-style-type: none"> db2GetSnapshot API - Get a snapshot of the database manager operational status db2MonitorSwitches API - Get or update the monitor switch settings

Table 3. COBOL Sample Programs with DB2 APIs

Sample Program	Included APIs
checkerr.cbl	<ul style="list-style-type: none"> sqlaintp API - Get error message sqlogstt API - Get the SQLSTATE message
client.cbl	<ul style="list-style-type: none"> sqlqryc API - Query client connection settings sqlesetc API - Set client connection settings
db_udcs.cbl	<ul style="list-style-type: none"> sqleatin API - Attach to instance sqlcrea API - Create database sqledrpd API - Drop database
dbcatt.cbl	<ul style="list-style-type: none"> sqlcadb API - Catalog a database in the system database directory db2DbDirCloseScan API - End a system or local database directory scan db2DbDirGetNextEntry API - Get the next system or local database directory entry db2DbDirOpenScan API - Start a system or local database directory scan sqluncd API - Uncatalog a database from the system database directory

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

Sample Program	Included APIs
dbcmt.cbl	<ul style="list-style-type: none"> • sqledcgd API - Change a database comment in the system or local database directory • db2DbDirCloseScan API - End a system or local database directory scan • db2DbDirGetNextEntry API - Get the next system or local database directory entry • db2DbDirOpenScan API - Start a system or local database directory scan • sqleisig API - Install signal handler
dbinst.cbl	<ul style="list-style-type: none"> • sqleatcp API - Attach to instance and change password • sqleatin API - Attach to instance • sqledtin API - Detach from instance • sqlegins API - Get current instance
dbstat.sqb	<ul style="list-style-type: none"> • db2Reorg API - Reorganize an index or a table • db2Runstats API - Update statistics about the characteristics of a table and associated indexes
dcscat.cbl	<ul style="list-style-type: none"> • sqlegdad API - Catalog a database in the database connection services (DCS) directory • sqlegdcl API - End a database connection services (DCS) directory scan • sqlegdel API - Uncatalog a database from the database connection services (DCS) directory • sqlegdge API - Get a specific entry in the database connection services (DCS) directory • sqlegdgt API - Get database connection services (DCS) directory entries • sqlegdsc API - Start a database connection services (DCS) directory scan
ebcdicdb.cbl	<ul style="list-style-type: none"> • sqleatin API - Attach to instance • sqlecrea API - Create database • sqledrpd API - Drop database
expsamp.sqb	<ul style="list-style-type: none"> • db2Export API - Export data from a database • db2Import API - Import data into a table, hierarchy, nickname or view
impexp.sqb	<ul style="list-style-type: none"> • db2Export API - Export data from a database • db2Import API - Import data into a table, hierarchy, nickname or view
loadqry.sqb	<ul style="list-style-type: none"> • db2LoadQuery API - Get the status of a load operation
migrate.cbl	<ul style="list-style-type: none"> • sqlemgdb API - Migrate previous version of DB2 database to current version
nodecat.cbl	<ul style="list-style-type: none"> • sqlectnd API - Catalog an entry in the node directory • sqlencls API - End a node directory scan • sqlengne API - Get the next node directory entry • sqlenops API - Start a node directory scan • sqleuncn API - Uncatalog an entry from the node directory

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

Sample Program	Included APIs
rebind.sqb	<ul style="list-style-type: none"> • sqlarbind API - Rebind package
tabscont.sqb	<ul style="list-style-type: none"> • sqlbctcq API - Close a table space container query • sqlbftcq API - Fetch the query data for rows in a table space container • sqlbotcq API - Open a table space container query • sqlbtcq API - Get the query data for all table space containers • sqlefmem API - Free the memory allocated by the sqlbctcq and sqlbmtsq APIs
tabspace.sqb	<ul style="list-style-type: none"> • sqlbctsq API - Close a table space query • sqlbftpq API - Fetch the query data for rows in a table space • sqlbgtss API - Get table space usage statistics • sqlbmtsq API - Get the query data for all table spaces • sqlbotsq API - Open a table space query • sqlbstpq API - Get information about a single table space • sqlefmem API - Free the memory allocated by the sqlbctcq and sqlbmtsq APIs
tload.sqb	<ul style="list-style-type: none"> • db2Export API - Export data from a database • sqluvqdp API - Quiesce table spaces for a table
tspace.sqb	<ul style="list-style-type: none"> • sqlbctcq API - Close a table space container query • sqlbctsq API - Close a table space query • sqlbftcq API - Fetch the query data for rows in a table space container • sqlbftpq API - Fetch the query data for rows in a table space • sqlbgtss API - Get table space usage statistics • sqlbmtsq API - Get the query data for all table spaces • sqlbotcq API - Open a table space container query • sqlbotsq API - Open a table space query • sqlbstpq API - Get information about a single table space • sqlbstsc API - Set table space containers • sqlbtcq API - Get the query data for all table space containers • sqlefmem API - Free the memory allocated by the sqlbctcq and sqlbmtsq APIs
setact.cbl	<ul style="list-style-type: none"> • sqlesact API - Set accounting string

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related tasks:

- “Building applications in C or C++ using the sample build script (UNIX)” in *Developing Embedded SQL Applications*
- “Building embedded SQL and DB2 API applications in C or C++ with configuration files (AIX)” in *Developing Embedded SQL Applications*
- “Building IBM COBOL applications on AIX” in *Developing Embedded SQL Applications*

- “Building UNIX Micro Focus COBOL applications” in *Developing Embedded SQL Applications*

Related reference:

- “COBOL samples” in *Samples Topics*
- “HP-UX C application compile and link options” in *Developing Embedded SQL Applications*
- “AIX IBM COBOL application compile and link options” in *Developing Embedded SQL Applications*
- “AIX Micro Focus COBOL application compile and link options” in *Developing Embedded SQL Applications*
- “C samples” in *Samples Topics*
- “AIX C embedded SQL and DB2 API applications compile and link options” in *Developing Embedded SQL Applications*
- “AIX C++ embedded SQL and DB2 administrative API applications compile and link options” in *Developing Embedded SQL Applications*
- “HP-UX C++ application compile and link options” in *Developing Embedded SQL Applications*
- “Include files for DB2 API applications” on page 19
- “HP-UX Micro Focus COBOL application compile and link options” in *Developing Embedded SQL Applications*
- “Include files for C and C++ embedded SQL applications” in *Developing Embedded SQL Applications*
- “Include files for COBOL embedded SQL applications” in *Developing Embedded SQL Applications*
- “Linux C application compile and link options” in *Developing Embedded SQL Applications*
- “Linux C++ application compile and link options” in *Developing Embedded SQL Applications*
- “Linux Micro Focus COBOL application compile and link options” in *Developing Embedded SQL Applications*
- “REXX samples” in *Samples Topics*
- “Solaris C application compile and link options” in *Developing Embedded SQL Applications*
- “Solaris C++ application compile and link options” in *Developing Embedded SQL Applications*
- “Solaris Micro Focus COBOL application compile and link options” in *Developing Embedded SQL Applications*

Include files for DB2 API applications

The include files that are intended to be used in your C, C++, COBOL and FORTRAN applications to call DB2 APIs are described below:

C and C++ include files

DB2APIDF (db2ApiDf.h)

This file defines structures, constants, and prototypes for almost all DB2 APIs whose names start with 'db2'.

DB2AUCFG (db2AuCfg.h)

This file defines structures, constants, and prototypes for the DB2 APIs, db2AutoConfig and db2AutoConfigFreeMemory.

DB2SECPLUGIN (db2secPlugin.h)

This file defines structures, constants, and prototypes for APIs used to develop customized security plug-ins for authentication and group membership lookup purposes.

SQL (sql.h) This file includes language-specific prototypes for the binder, precompiler, and error message retrieval APIs. It also defines system constants.

SQLAPREP (sqlaprep.h)

This file contains definitions required to write your own precompiler.

SQLENV (sqlenv.h)

This file defines language-specific calls for the database environment APIs, and the structures, constants, and return codes for those interfaces.

SQLMON (sqlmon.h)

This file defines language-specific calls for the database system monitor APIs, and the structures, constants, and return codes for those interfaces.

SQLUTIL (sqlutil.h)

This file defines the language-specific calls for the utility APIs, and the structures, constants, and codes required for those interfaces.

SQLUVEND (sqluvend.h)

This file defines structures, constants, and prototypes for the APIs to be used by the storage management vendors.

SQLXA (sqlxa.h)

This file contains function prototypes and constants used by applications that use the X/Open XA Interface.

COBOL include files

SQL (sql.cbl) This file includes language-specific prototypes for the binder, precompiler, and error message retrieval APIs. It also defines system constants.

SQLAPREP (sqlaprep.cbl)

This file contains definitions required to write your own precompiler.

SQLENV (sqlenv.cbl)

This file defines language-specific calls for the database environment APIs, and the structures, constants, and return codes for those interfaces.

SQLMON (sqlmon.cbl)

This file defines language-specific calls for the database system monitor APIs, and the structures, constants, and return codes for those interfaces.

SQLMONCT (sqlmonct.cbl)

This file contains constant definitions and local data structure definitions required to call the Database System Monitor APIs.

SQLUTIL (sqlutil.cbl)

This file defines the language-specific calls for the utility APIs, and the structures, constants, and codes required for those interfaces.

FORTRAN include files

SQL (sql.f)

This file includes language-specific prototypes for the binder, precompiler, and error message retrieval APIs. It also defines system constants.

SQLAPREP (sqlaprep.f)

This file contains definitions required to write your own precompiler.

SQLENV (sqlenv.f)

This file defines language-specific calls for the database environment APIs, and the structures, constants, and return codes for those interfaces.

SQLMON (sqlmon.f)

This file defines language-specific calls for the database system monitor APIs, and the structures, constants, and return codes for those interfaces.

SQLUTIL (sqlutil.f)

This file defines the language-specific calls for the utility APIs, and the structures, constants, and codes required for those interfaces.

Related reference:

- “DB2 APIs” on page 1

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, the scope can be the collection of all logical database partition servers defined in the node configuration file (`db2nodes.cfg`) or the database partition from which the API is called.

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.

None means that no database connection is required in order for the API to work successfully. *Establishes a connection* means that the API will establish a connection to the database when the API is called.

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`

How the API descriptions are organized

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.

Related reference:

- “DB2 APIs” on page 1

db2AddContact - Add a contact to whom notification messages can be sent

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`

API and data structure syntax:

```
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;
```

db2AddContact 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.

db2AddContactData data structure parameters:

piUserid

Input. The user name.

piPassword

Input. The password for the user ID specified by parameter `piUserid`.

piName

Input. The contact name.

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

db2AddContact - Add a contact to whom notification messages can be sent

- 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.

Usage notes:

This API is not supported on UNIX[®] and Linux[®]. However, you can access the same functionality through the SQL interface.

Related tasks:

- “Enabling health alert notification” in *System Monitor Guide and Reference*

Related reference:

- “ADD CONTACT command” in *Command Reference*
- “db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent” on page 77
- “db2GetContacts - Get the list of contacts to whom notification messages can be sent” on page 105
- “db2UpdateContact - Update the attributes of a contact” on page 336
- “SQLCA ” on page 526
- “contact_host - Location of contact list configuration parameter” in *Performance Guide*
- “db2AddContactGroup - Add a contact group to whom notification messages can be sent” on page 26
- “ADD CONTACT command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

db2AddContactGroup - Add a contact group to whom notification messages can be sent

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

API and data structure syntax:

```
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;
```

db2AddContactGroup 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.

db2AddContactGroupData data structure parameters:

piUserid

Input. The user name.

db2AddContactGroup - Add a contact group to whom notification messages can be sent

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.

db2ContactTypeData data structure parameters:

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.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

Related reference:

- "ADD CONTACTGROUP command" in *Command Reference*
- "db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent" on page 79
- "db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent" on page 101
- "db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent" on page 103
- "contact_host - Location of contact list configuration parameter" in *Performance Guide*
- "db2UpdateContactGroup - Update the attributes of a contact group" on page 338
- "SQLCA " on page 526
- "db2AddContact - Add a contact to whom notification messages can be sent" on page 24
- "ADD CONTACTGROUP command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

db2AddSnapshotRequest - Add a snapshot request

This API prepares the snapshot request stream for db2GetSnapshotSize and db2GetSnapshot.

Scope:

Prepares the snapshot request stream for the db2GetSnapshotSize and db2GetSnapshot APIs. The output (a snapshot request that is generated by the db2AddSnapshotRequest API) is passed to the db2GetSnapshotSize and db2GetSnapshot APIs. A snapshot request contains the snapshot request type and the identification information.

Authorization:

None.

Required connection:

None.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2AddSnapshotRequest (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2AddSnapshotRqstData
{
    void *pioRequestData;
    db2UInt32 iRequestType;
    db2int32 iRequestFlags;
    db2UInt32 iQualType;
    void *piQualData;
} db2AddSnapshotRqstData;

SQL_API_RC SQL_API_FN
db2gAddSnapshotRequest (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gAddSnapshotRqstData
{
    void *pioRequestData;
    db2UInt32 iRequestType;
    db2int32 iRequestFlags;
    db2UInt32 iQualType;
    void *piQualData;
    db2UInt32 iQualDataLen;
} db2gAddSnapshotRqstData;
```

db2AddSnapshotRequest API parameters:

versionNumber

Input. Specifies the version and release level of the structure passed as the

db2AddSnapshotRequest - Add a snapshot request

second parameter pParmStruct. To use the structure db2AddSnapshotData as described above, specify db2Versio910. If you want to use a different version of this structure, check the db2ApiDf header file in the include directory for the complete list of supported versions. Ensure that you use the version of the db2AddSnapshotRequestData structure that corresponds to the version number that you specify.

pParmStruct

Input and/or output. A pointer to the db2AddSnapshotRequestData structure.

pSqlca

Output. A pointer to the sqlca structure.

db2AddSnapshotRqstData data structure parameters:

pioRequestData

Input/output. The request data to be constructed by the db2AddSnapshotRequest API. Initially, this parameter is set to NULL. The memory required for pioRequestData will be allocated by the db2AddSnapshotRequest API. You should free pioRequestData when its usage ends (for example, after the db2GetSnapshot API call).

iRequestType

Input. Snapshot request type, for example, SQLMA_DB2.

iRequestFlags

Input. Bit mapped action flags, the values are SQLM_INSTREAM_ADD_REQUEST, SQLM_INSTREAM_ADD_QUAL or SQLM_INSTREAM_ADD_REQQUAL. If iRequestFlags is not set by the caller:

- if iRequestType is set, iRequestFlags bit SQLM_INSTREAM_ADD_REQUEST is turned on by the API.
- if piQualifierData pointer is not null, SQLM_INSTREAM_ADD_QUAL is turned on by the API.

Upon API call, iRequestType, iQualifierType, iRequestFlags and piQualifierData are reset to 0.

iQualType

Input. Type of the qualifier attached to the snapshot request, for example, SQLM_INSTREAM_ELM_DBNAME.

piQualData

Input. Data describing the qualifier. This is a pointer to a null-terminated string.

db2gAddSnapshotRqstData data structure specific parameters:

iQualDataLen

Input. Length of the qualifier data in the piQualData parameter.

Related reference:

- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API” on page 117
- “SQLCA ” on page 526

db2AdminMsgWrite - Write log messages for administration and replication function

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2AdminMsgWrite (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef struct db2AdminMsgWriteStruct
{
    db2UInt32 iMsgType;
    db2UInt32 iComponent;
    db2UInt32 iFunction;
    db2UInt32 iProbeID;
    char *piData_title;
    void *piData;
    db2UInt32 iDataLen;
    db2UInt32 iError_type;
} db2AdminMsgWriteStruct;
```

db2AdminMsgWrite 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.

db2AdminMsgWriteStruct data structure parameters:

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.

db2AdminMsgWrite - Write log messages for administration and replication function

iProbeID

Input. Specify the numeric probe point. Numeric probe point is a unique internal identifier that is used to locate the point in the source code that reported the message.

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:

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 526

db2ArchiveLog - Archive the active log file

Closes and truncates the active log file for a recoverable database. If user exit is enabled, it also 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2ArchiveLog (
    db2UInt32 versionNumber,
    void * pDB2ArchiveLogStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ArchiveLogStruct
{
    char *piDatabaseAlias;
    char *piUserName;
    char *piPassword;
    db2UInt16 iAllNodeFlag;
    db2UInt16 iNumNodes;
    SQL_PDB_NODE_TYPE *piNodeList;
    db2UInt32 iOptions;
} db2ArchiveLogStruct;

SQL_API_RC SQL_API_FN
db2gArchiveLog (
    db2UInt32 versionNumber,
    void * pDB2ArchiveLogStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gArchiveLogStruct
{
    db2UInt32 iAliasLen;
    db2UInt32 iUserNameLen;
    db2UInt32 iPasswordLen;
    char *piDatabaseAlias;
    char *piUserName;
    char *piPassword;
    db2UInt16 iAllNodeFlag;
    db2UInt16 iNumNodes;
    SQL_PDB_NODE_TYPE *piNodeList;
    db2UInt32 iOptions;
} db2gArchiveLogStruct;
```

db2ArchiveLog API parameters:

versionNumber

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.

db2ArchiveLogStruct data structure parameters:

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

Applicable to partitioned database environment 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

Partitioned database environment only. Input. Specifies the number of nodes in the piNodeList array.

piNodeList

Partitioned database environment only. Input. A pointer to an array of node numbers against which to apply the archive log operation.

iOptions

Input. Reserved for future use.

db2gArchiveLogStruct data structure specific parameters:

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.

db2ArchiveLog - Archive the active log file

iPasswordLen

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

Related reference:

- “ARCHIVE LOG command” in *Command Reference*
- “SQLCA ” on page 526

db2AutoConfig - Access the Configuration Advisor

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

Scope:

In a partitioned database environment, database recommendations are applied by default on all database partitions. DB2_SG_APPLY_ON_ONE_NODE flag for the db2AutoConfigInterface data structure's iApply parameter forces the changes to be limited to the coordinator partition only. Note that the bufferpool changes are always (DB2_SG_APPLY_ON_ONE_NODE does not matter for bufferpool recommendations) applied to the system catalogs, thus, all database partitions are affected.

Authorization:

sysadm

Required connection:

Database

API include file:

db2AuCfg.h

API and data structure syntax:

```
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+1];
    char iDbAlias[SQL_ALIAS_SZ+1];
    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 db2CfgParam * pConfigs;
    void * pDataArea;
} db2ConfigValues;
```

db2AutoConfig - Access the Configuration Advisor

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

typedef struct {
    db2Uint32 numElements;
    db2AutoConfigNameElement * pElements;
} db2AutoConfigNameArray;
typedef db2AutoConfigNameArray db2BpValues;

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

db2AutoConfig 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.

db2AutoConfigInterface data structure parameters:

iProductID

Input. Specifies a unique product identifier. Valid values for the iProductID parameter (defined in db2AuCfg.h, located in the include directory) are:

- DB2_SG_PID_DEFAULT
- DB2_SG_PID_WEBSPHERE_COMMERCE_SUITE
- DB2_SG_PID_SAP
- DB2_SG_PID_WEBSPHERE_ADVANCED_SERVER
- DB2_SG_PID_SIEBEL
- DB2_SG_PID_PS_EPM
- DB2_SG_PID_PS_ONLINE
- DB2_SG_PID_PS_BATCH
- DB2_SG_PID_PS
- DB2_SG_PID_LOTUS_DOMINO
- DB2_SG_PID_CONTENT_MANAGER

iProductVersion

Input. A 16 byte string specifying the product version.

iDbAlias

Input. A string specifying a database alias.

iApply

Input. Updates the configuration automatically. Valid values for the iApply parameter (defined in db2AuCfg.h, located in the include directory) are:

db2AutoConfig - Access the Configuration Advisor

DB2_SG_NOT_APPLY

Do not apply any recommendations

DB2_SG_APPLY

Apply all recommendations

DB2_SG_APPLY_DB

Apply only database (and bufferpool) recommendations

DB2_SG_APPLY_ON_ONE_NODE

Apply database recommendations (valid only with DB2_SG_APPLY and DB2_SG_APPLY_DB) on the current database partition only. By default the database recommendations are applied on all database partitions.

iParams

Input. Passes parameters into the advisor.

oResult

Output. Includes all results from the advisor.

db2AutoConfigElement data structure parameters:

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

value Input or output. Holds the data specified by the token.

db2AutoConfigArray data structure parameters:

numElements

Input or output. The number of array elements.

pElements

Input or output. A pointer to the element array.

db2ConfigValues data structure parameters:

numElements

Input or output. The number of array elements.

pConfigs

Input or output. A pointer to an array of db2CfgParam structure.

pDataArea

Input or output. A pointer to the data area containing the values of the configuration.

db2AutoConfigNameElement data structure parameters:

pName

Output. The name of the output buffer pool.

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

db2AutoConfigNameArray data structure parameters:

numElements

Input or output. The number of array elements.

pElements

Input or output. A pointer to the element array.

db2AutoConfig - Access the Configuration Advisor

db2AutoConfigOutput data structure parameters:

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 advisor. 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 advisor. 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 advisor.

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 advisor.

oDiagnostics

Output. Includes diagnostics from the advisor.

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 advisor. 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 advisor.

Usage notes:

To free the memory allocated by the db2AutoConfig API, call the db2AutoConfigFreeMemory API.

Related concepts:

- “Configuration parameters” in *Performance Guide*

Related tasks:

- “Configuring DB2 with configuration parameters” in *Performance Guide*
- “Changing node and database configuration files” in *Administration Guide: Implementation*

Related reference:

- “AUTOCONFIGURE command” in *Command Reference*
- “CREATE DATABASE command” in *Command Reference*
- “db2AutoConfigFreeMemory - Free the memory allocated by the db2AutoConfig API” on page 40
- “db2CfgSet - Set the database manager or database configuration parameters” on page 53
- “SQLCA ” on page 526
- “sqlfupd ” on page 567

db2AutoConfig - Access the Configuration Advisor

- “AUTOCONFIGURE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2CfgGet - Get the database manager or database configuration parameters” on page 49

Related samples:

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

db2AutoConfigFreeMemory - Free the memory allocated by the db2AutoConfig API

Frees the memory allocated by the db2AutoConfig API.

Authorization:

sysadm

Required connection:

Database

API include file:

db2AuCfg.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2AutoConfigFreeMemory(
    db2UInt32 db2VersionNumber,
    void * pAutoConfigInterface,
    struct sqlca * pSqlca);
```

db2AutoConfigFreeMemory 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:

- “db2AutoConfig - Access the Configuration Advisor” on page 35
- “SQLCA ” on page 526

Related samples:

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

db2Backup - Back up a database or table space

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

API and data structure syntax:

```
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;
    db2UInt32 iUtilImpactPriority;
    char *piComprLibrary;
    void *piComprOptions;
    db2UInt32 iComprOptionsSize;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
    char **tablespaces;
    db2UInt32 numTablespaces;
} db2TablespaceStruct;
```

db2Backup - Back up a database or table space

```
typedef SQL_STRUCTURE db2MediaListStruct
{
    char                **locations;
    db2UInt32 numLocations;
    char locationType;
} db2MediaListStruct;

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;
    db2UInt32 iUtilImpactPriority;
    char *piComprLibrary;
    db2UInt32 iComprLibraryLen;
    void *piComprOptions;
    db2UInt32 iComprOptionsSize;
} 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;
```

db2Backup API parameters:

versionNumber

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

db2Backup - Back up a database or table space

pDB2BackupStruct

Input. A pointer to the db2BackupStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2BackupStruct data structure parameters:

piDBAlias

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

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.

oTimestamp

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

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 parameter (defined in sqlutil header file, located in the include directory) 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.

piPassword

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

piVendorOptions

Input. Used to pass information from the application to the vendor

db2Backup - Back up a database or table space

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 header file, located in the include directory) 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 4 KB allocation units (pages). Minimum is 8 units.

iNumBuffers

Input. Specifies number of backup buffers to be used. Minimum is 2. Maximum is limited by memory.

iParallelism

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

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 header file, located in the include directory) 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.

DB2BACKUP_COMPRESS

Specifies that the backup should be compressed.

DB2BACKUP_INCLUDE_COMPR_LIB

Specifies that the library used for compressing the backup should be included in the backup image.

DB2BACKUP_EXCLUDE_COMPR_LIB

Specifies that the library used for compressing the backup should be not included in the backup image.

DB2BACKUP_INCLUDE_LOGS

Specifies that the backup image should also include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup or a multi-partition backup.

DB2BACKUP_EXCLUDE_LOGS

Specifies that the backup image should not include any log files.

Note: When performing an offline backup operation, logs are excluded whether or not this option is specified.

iUtilImpactPriority

Input. Specifies the priority value to be used during a backup.

db2Backup - Back up a database or table space

- If this value is non-zero, the utility will run throttled. Otherwise, the utility will run unthrottled.
- If there are multiple concurrent utilities running, this parameter is used to determine a relative priority between the throttled tasks. For example, consider two concurrent backups, one with priority 2 and another with priority 4. Both will be throttled, but the one with priority 4 will be allotted more resources. Setting priorities to 2 and 4 is no different than setting them to 5 and 10 or 30 and 60. Priorities values are purely relative.

piComprLibrary

Input. Indicates the name of the external library to be used to perform compression of the backup image. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will use the default library for compression. If the specified library is not found, the backup will fail.

piComprOptions

Input. Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize

Input. A four-byte unsigned integer representing the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

db2TablespaceStruct data structure specific parameters:

tablespaces

Input. 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

Input. Number of entries in the tablespaces parameter.

db2MediaListStruct data structure parameters:

locations

Input. 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

Input. The number of entries in the locations parameter.

locationType

Input. A character indicating the media type. Valid values (defined in sqlutil header file, located in the include directory.) are:

SQLU_LOCAL_MEDIA

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

db2Backup - Back up a database or table space

SQLI_XBSA_MEDIA

XBSA interface.

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).

db2gBackupStruct data structure specific parameters:

iDBAliasLen

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

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).

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).

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.

iPasswordLen

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

iComprLibraryLen

Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in piComprLibrary. Set to zero if no library name is given.

db2Char data structure parameters:

pioData

A pointer to a character data buffer. If NULL, no data will be returned.

iLength

Input. The size of the pioData buffer.

oLength

Output. The number of valid characters of data in the pioData buffer.

Usage notes:

This function is exempt from all label-based access control (LBAC) rules. It backs up all data, even protected data. Also, the data in the backup itself is not protected by LBAC. Any user with the backup and a place in which to restore it can gain access to the data.

Related tasks:

- “Using backup” in *Data Recovery and High Availability Guide and Reference*

db2Backup - Back up a database or table space

Related reference:

- “SQLCA ” on page 526
- “BACKUP DATABASE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “BACKUP DATABASE command” in *Command Reference*
- “db2Rollforward - Roll forward a database” on page 296
- “db2Restore - Restore a database or table space” on page 284
- “db2Recover - Restore and roll forward a database” on page 265

Related samples:

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

db2CfgGet - Get the database manager or database 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

API and data structure syntax:

```
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;

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;
```

db2CfgGet - Get the database manager or database configuration parameters

```
    char *dbname;  
} db2gCfg;  
  
typedef SQL_STRUCTURE db2gCfgParam  
{  
    db2UInt32 token;  
    db2UInt32 ptrvalue_len;  
    char *ptrvalue;  
    db2UInt32 flags;  
} db2gCfgParam;
```

db2CfgGet 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.

db2Cfg data structure parameters:

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 Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

db2CfgReset

Reset to default values.

dbname

Input. The database name.

db2CfgParam data structure parameters:

token Input. The configuration parameter identifier.

db2CfgGet - Get the database manager or database configuration parameters

Valid entries and data types for the db2CfgParam token element are listed in "Configuration parameters summary".

ptrvalue

Output. The configuration parameter value.

flags Input. Provides specific information for each parameter in a request. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgParamAutomatic

Indicates whether the retrieved parameter has a value of automatic. To determine whether a given configuration parameter has been set to automatic, perform a boolean AND operation against the value returned by the flag and the db2CfgParamAutomatic keyword defined in db2ApiDf.h.

db2gCfg data structure specific parameters:

dbname_len

Input. The length in bytes of dbname parameter.

db2gCfgParam data structure specific parameters:

ptrvalue_len

Input. The length in bytes of ptrvalue parameter.

Related tasks:

- "Configuring DB2 with configuration parameters" in *Performance Guide*

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "DBCFCG administrative view – Retrieve database configuration parameter information" in *Administrative SQL Routines and Views*
- "DBMCFG administrative view – Retrieve database manager configuration parameter information" in *Administrative SQL Routines and Views*
- "Configuration parameters summary" in *Performance Guide*
- "db2CfgSet - Set the database manager or database configuration parameters" on page 53
- "SQLCA " on page 526
- "db2AutoConfig - Access the Configuration Advisor" on page 35

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++)"

db2CfgGet - Get the database manager or database configuration parameters

- “tscreate.sql -- How to create and drop buffer pools and table spaces (C++)”

db2CfgSet - Set the database manager or database 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 attachment to an instance is not required.

API include file:

db2ApiDf.h

API and data structure syntax:

```
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;

SQL_API_RC SQL_API_FN
db2gCfgSet (
    db2UInt32 versionNumber,
    void * pParmStruct,
```

db2CfgSet - Set the database manager or database configuration parameters

```
    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;
```

db2CfgSet 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.

db2Cfg data structure parameters:

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 Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

db2CfgReset

Reset to default values.

db2CfgSet - Set the database manager or database configuration parameters

dbname

Input. The database name.

db2CfgParam data structure parameters:

token Input. The configuration parameter identifier. Valid entries and data types for the db2CfgParam token element are listed in "Configuration parameters summary".

ptrvalue

Output. The configuration parameter value.

flags Input. Provides specific information for each parameter in a request. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgParamAutomatic

Indicates whether the retrieved parameter has a value of automatic. To determine whether a given configuration parameter has been set to automatic, perform a boolean AND operation against the value returned by the flag and the db2CfgParamAutomatic keyword defined in db2ApiDf.h.

db2gCfg data structure specific parameters:

dbname_len

Input. The length in bytes of dbname parameter.

db2gCfgParam data structure specific parameters:

ptrvalue_len

Input. The length in bytes of ptrvalue parameter.

Related concepts:

- "Configuration parameters" in *Performance Guide*

Related tasks:

- "Configuring DB2 with configuration parameters" in *Performance Guide*
- "Changing node and database configuration files" in *Administration Guide: Implementation*

Related reference:

- "AUTOCONFIGURE command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "Configuration parameters summary" in *Performance Guide*
- "db2CfgGet - Get the database manager or database configuration parameters" on page 49
- "SQLCA " on page 526

db2CfgSet - Set the database manager or database configuration parameters

- “RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2AutoConfig - Access the Configuration Advisor” on page 35

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 the monitor stream to the pre-version 6 format

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2ConvMonStream (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ConvMonStreamData
{
    void *poTarget;
    struct sqlm_header_info *piSource;
    db2UInt32 iTargetType;
    db2UInt32 iTargetSize;
    db2UInt32 iSourceType;
} db2ConvMonStreamData;

SQL_API_RC SQL_API_FN
db2gConvMonStream (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2ConvMonStream 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 db2ConvMonStreamData structure.

pSqlca

Output. A pointer to the sqlca structure.

db2ConvMonStreamData data structure parameters:

db2ConvMonStream - Convert the monitor stream to the pre-version 6 format

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_SIZ` 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
<code>SQLM_ELM_APPL</code>	<code>sqlm_appl</code>
<code>SQLM_ELM_APPL_INFO</code>	<code>sqlm_applinfo</code>
<code>SQLM_ELM_DB2</code>	<code>sqlm_db2</code>
<code>SQLM_ELM_FCM</code>	<code>sqlm_fcm</code>
<code>SQLM_ELM_FCM_NODE</code>	<code>sqlm_fcm_node</code>
<code>SQLM_ELM_DBASE</code>	<code>sqlm_dbase</code>
<code>SQLM_ELM_TABLE_LIST</code>	<code>sqlm_table_header</code>
<code>SQLM_ELM_TABLE</code>	<code>sqlm_table</code>
<code>SQLM_ELM_DB_LOCK_LIST</code>	<code>sqlm_dbase_lock</code>
<code>SQLM_ELM_APPL_LOCK_LIST</code>	<code>sqlm_appl_lock</code>
<code>SQLM_ELM_LOCK</code>	<code>sqlm_lock</code>
<code>SQLM_ELM_STMT</code>	<code>sqlm_stmt</code>
<code>SQLM_ELM_SUBSECTION</code>	<code>sqlm_subsection</code>
<code>SQLM_ELM_TABLESPACE_LIST</code>	<code>sqlm_tablespace_header</code>
<code>SQLM_ELM_TABLESPACE</code>	<code>sqlm_tablespace</code>
<code>SQLM_ELM_ROLLFORWARD</code>	<code>sqlm_rollfwd_info</code>
<code>SQLM_ELM_BUFFERPOOL</code>	<code>sqlm_bufferpool</code>
<code>SQLM_ELM_LOCK_WAIT</code>	<code>sqlm_lockwait</code>

db2ConvMonStream - Convert the monitor stream to the pre-version 6 format

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

Snapshot variable datastream type	Structure
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
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 526

db2DatabasePing - Ping the database to test network response time

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DatabasePing (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DatabasePingStruct
{
    char iDbAlias[SQL_ALIAS_SZ + 1];
    db2int32 RequestPacketSz;
    db2int32 ResponsePacketSz;
    db2UInt16 iNumIterations;
    db2UInt32 *poElapsedTime;
} db2DatabasePingStruct;

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 + 1];
    db2int32 RequestPacketSz;
    db2int32 ResponsePacketSz;
    db2UInt16 iNumIterations;
    db2UInt32 *poElapsedTime;
} db2gDatabasePingStruct;
```

db2DatabasePing API parameters:

versionNumber

Input. Specifies the version and release of the DB2 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.

db2DatabasePing - Ping the database to test network response time

db2DatabasePingStruct data structure parameters:

iDbAlias

Input. Database alias name. Reserved for future use. If a value is provided, it is ignored.

RequestPacketSz

Input. Size of the packet (in bytes) to be sent to the server. The size must be between 0 and 32767 inclusive. This parameter is only valid on servers running DB2 Universal Database (UDB) for Linux, UNIX and Windows Version 8 or higher, or DB2 UDB for z/OS Version 8 or higher.

ResponsePacketSz

Input. Size of the packet (in bytes) to be returned back to client. The size must be between 0 and 32767 inclusive. This parameter is only valid on servers running DB2 UDB for Linux, UNIX and Windows Version 8 or higher, or DB2 UDB for z/OS Version 8 or higher.

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.

db2gDatabasePingStruct data structure specific parameters:

iDbAliasLength

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

Usage notes:

This API will not work when it is used from a DB2 Universal Database Version 7 client through a DB2 Connect Version 8 to a connected DB2 host database server.

Related reference:

- "SQLCA " on page 526
- "PING command" in *Command Reference*

db2DatabaseQuiesce - Quiesce the database

Forces all users off the database, immediately rolls back all active transactions or waits for them to complete their current units of work within the number of minutes specified (if they cannot be completed within the specified number of minutes, the operation will fail), 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 by users with appropriate authority. 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DatabaseQuiesce (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbQuiesceStruct
{
    char *piDatabaseName;
    db2UInt32 iImmediate;
    db2UInt32 iForce;
    db2UInt32 iTimeout;
} db2DbQuiesceStruct;

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;
    db2UInt32 iTimeout;
} db2gDbQuiesceStruct;
```

db2DatabaseQuiesce 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.

db2DbQuiesceStruct data structure parameters:**piDatabaseName**

Input. The database name.

iImmediate

Input. Valid values are:

TRUE=1

Force the applications immediately.

FALSE=0

Deferred force. Applications will wait the number of minutes specified by iTimeout parameter to let their current units of work be completed, and then will terminate. If this deferred force cannot be completed within the number of minutes specified by iTimeout parameter, the quiesce operation will fail.

iForce Input. Reserved for future use.

iTimeout

Input. Specifies the time, in minutes, to wait for applications to commit the current unit of work. If iTimeout is not specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the start_stop_timeout database manager configuration parameter will be used.

db2gDbQuiesceStruct data structure specific parameters:**iDatabaseNameLen**

Input. Specifies the length in bytes of piDatabaseName.

Related reference:

- “QUIESCE command” in *Command Reference*
- “db2DatabaseUnquiesce - Unquiesce database” on page 67
- “SQLCA ” on page 526
- “QUIESCE DATABASE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DatabaseRestart (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef struct db2RestartDbStruct
{
    char *piDatabaseName;
    char *piUserId;
    char *piPassword;
    char *piTablespaceNames;
    db2Int32 iOption;
} db2RestartDbStruct;

SQL_API_RC SQL_API_FN
db2gDatabaseRestart (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef struct db2gRestartDbStruct
{
    db2UInt32 iDatabaseNameLen;
    db2UInt32 iUserIdLen;
    db2UInt32 iPasswordLen;
    db2UInt32 iTablespaceNamesLen;
    char *piDatabaseName;
    char *piUserId;
    char *piPassword;
    char *piTablespaceNames;
} db2gRestartDbStruct;
```

db2DatabaseRestart 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 db2RestartDbStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2RestartDbStruct data structure parameters:**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.

db2gRestartDbStruct data structure specific parameters:**iDatabaseNameLen**

Input. Length in bytes of piDatabaseName parameter.

iUserIdLen

Input. Length in bytes of piUserId parameter.

iPasswordLen

Input. Length in bytes of piPassword parameter.

iTablespaceNamesLen

Input. Length in bytes of piTablespaceNames parameter.

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.

db2DatabaseRestart - Restart database

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.

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.

Related reference:

- “RESTART DATABASE command” in *Command Reference*
- “SQLCA ” on page 526

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++)”

db2DatabaseUnquiesce - Unquiesce database

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DatabaseUnquiesce (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

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

```
SQL_API_RC SQL_API_FN
db2gDatabaseUnquiesce (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2gDbUnquiesceStruct
{
    db2UInt32 iDatabaseNameLen;
    char *piDatabaseName;
} db2gDbUnquiesceStruct;
```

db2DatabaseUnquiesce 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.

db2DbUnquiesceStruct data structure parameters:

piDatabaseName

Input. The database name.

db2DatabaseUnquiesce - Unquiesce database

db2gDbUnquiesceStruct data structure specific parameters:

iDatabaseNameLen

Input. Specifies the length in bytes of piDatabaseName.

Related reference:

- “UNQUIESCE command” in *Command Reference*
- “db2DatabaseQuiesce - Quiesce the database” on page 62
- “SQLCA ” on page 526
- “UNQUIESCE DATABASE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

db2DbDirCloseScan - End a system or local database directory scan

Frees the resources allocated by db2DbDirOpenScan.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DbDirCloseScan (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2DbDirCloseScanStruct
{
    db2UInt16 iHandle;
} db2DbDirCloseScanStruct;
```

```
SQL_API_RC SQL_API_FN
db2gDbDirCloseScan (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2gDbDirCloseScanStruct
{
    db2UInt16 iHandle;
} db2gDbDirCloseScanStruct;
```

db2DbDirCloseScan 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 db2DbDirCloseScanStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2DbDirCloseScanStruct data structure parameters:

iHandle

Input. Identifier returned from the associated db2DbDirOpenScan API.

Related reference:

- “db2DbDirGetNextEntry - Get the next system or local database directory entry” on page 71
- “db2DbDirOpenScan - Start a system or local database directory scan” on page 75

db2DbDirCloseScan - End a system or local database directory scan

- “SQLCA ” on page 526

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++)”

db2DbDirGetNextEntry - Get the next system or local database directory entry

Returns the next entry in the system database directory or the local database directory copy returned by db2DbDirOpenScan. Subsequent calls to this API return additional entries.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DbDirGetNextEntry (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbDirNextEntryStruct
{
    db2UInt16 iHandle;
    struct db2DbDirInfo *poDbDirEntry;
} db2DbDirNextEntryStruct;

SQL_STRUCTURE db2DbDirInfo
{
    _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;
    _SQLOLDCHAR althostname[SQL_HOSTNAME_SZ];
    _SQLOLDCHAR altportnumber[SQL_SERVICE_NAME_SZ];
};

SQL_API_RC SQL_API_FN
db2gDbDirGetNextEntry (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbDirNextEntryStruct
{
    db2UInt16 iHandle;
    struct db2DbDirInfo *poDbDirEntry;
} db2gDbDirNextEntryStruct;
```

db2DbDirGetNextEntry - Get the next system or local database directory entry

db2DbDirGetNextEntry 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 db2DbDirGetNextEntryStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2DbDirNextEntryStruct data structure parameters:

iHandle

Input. Identifier returned from the associated db2DbDirOpenScan API.

poDbDirEntry

Output. A pointer to a db2DbDirInfo structure. The space for the directory data is allocated by the API, and a pointer to that space is returned to the caller.

db2DbDirInfo data structure parameters:

alias An alternate database name.

dbname

The name of the database.

drive 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, this parameter is CHAR(12).

intname

A token identifying the database subdirectory. This field is returned only if the local database directory is opened for scan.

nodename

The name of the node where the database is located. This field is returned only if the cataloged database is a remote database.

dbtype

Database manager release information.

comment

The comment associated with the database.

com_codepage

The code page of the comment. Not used.

type Entry type. Valid values 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).

db2DbDirGetNextEntry - Get the next system or local database directory entry

SQL_DCE

Database resides in DCE directories.

authentication

Authentication type. Valid values 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.

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.

SQL_AUTHENTICATION_DATAENC

Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption.

SQL_AUTHENTICATION_GSSPLUGIN

Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

glbdbname

The global name of the target database in the global (DCE) directory, if the entry is of type SQL_DCE.

dceprincipal

The principal name if the authentication is of type DCE or KERBEROS.

cat_nodenum

Catalog node number.

nodenum

Node number.

althostname

The hostname or IP address of the alternate server where the database is reconnected at failover time.

altportnumber

The port number of the alternate server where the database is reconnected at failover time.

Usage notes:

db2DbDirGetNextEntry - Get the next system or local database directory entry

All fields of the directory entry information buffer are padded to the right with blanks.

A subsequent db2DbDirGetNextEntry 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 db2DbDirGetNextEntry is called.

The count value returned by the db2DbDirOpenScan API can be used to scan through the entire directory by issuing db2DbDirGetNextEntry calls, one at a time, until the number of scans equals the count of entries.

Related reference:

- "LIST DATABASE DIRECTORY command" in *Command Reference*
- "db2DbDirCloseScan - End a system or local database directory scan" on page 69
- "db2DbDirOpenScan - Start a system or local database directory scan" on page 75
- "SQLCA " on page 526

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++)"

db2DbDirOpenScan - Start a system or local 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 the `db2DbDirGetNextEntry` API to advance through the database directory, examining information about the database entries. Close the scan using the `db2DbDirCloseScan` API. This removes the copy of the directory from memory.

Authorization:

None

Required connection:

None

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2DbDirOpenScan (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbDirOpenScanStruct
{
    char *piPath;
    db2UInt16 oHandle;
    db2UInt16 oNumEntries;
} db2DbDirOpenScanStruct;

SQL_API_RC SQL_API_FN
db2gDbDirOpenScan (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbDirOpenScanStruct
{
    db2UInt32 iPath_len;
    char *piPath;
    db2UInt16 oHandle;
    db2UInt16 oNumEntries;
} db2gDbDirOpenScanStruct;
```

db2DbDirOpenScan 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 `db2DbDirOpenScanStruct` structure.

pSqlca

Output. A pointer to the `sqlca` structure.

db2DbDirOpenScan - Start a system or local database directory scan

db2DbDirOpenScanStruct data structure parameters:

piPath 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.

oHandle

Output. A 2-byte area for the returned identifier. This identifier must be passed to the db2DbDirGetNextEntry API for scanning the database entries, and to the db2DbDirCloseScan API to release the resources.

oNumEntries

Output. A 2-byte area where the number of directory entries is returned.

db2gDbDirOpenScanStruct data structure specific parameters:

iPath_len

Input. The length in bytes of the piPath parameter.

Usage notes:

Storage allocated by this API is freed by the db2DbDirCloseScan API.

Multiple db2DbDirOpenScan 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:

- “db2DbDirCloseScan - End a system or local database directory scan” on page 69
- “db2DbDirGetNextEntry - Get the next system or local database directory entry” on page 71
- “SQLCA ” on page 526

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++)”

db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent

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

API and data structure syntax:

```
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;
```

db2DropContact 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.

db2DropContactData data structure parameters:

piUserid

Input. The user name.

piPassword

Input. The password for piUserid.

piName

Input. The name of the contact to be dropped.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent

Related reference:

- “DROP CONTACT command” in *Command Reference*
- “db2AddContact - Add a contact to whom notification messages can be sent” on page 24
- “db2GetContacts - Get the list of contacts to whom notification messages can be sent” on page 105
- “db2UpdateContact - Update the attributes of a contact” on page 336
- “contact_host - Location of contact list configuration parameter” in *Performance Guide*
- “SQLCA ” on page 526
- “DROP CONTACT command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent” on page 79

db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent

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

API and data structure syntax:

```
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;
```

db2DropContactGroup 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.

db2DropContactData data structure parameters:

piUserid

Input. The user name.

piPassword

Input. The password for piUserid.

piName

Input. The name of the contact to be dropped.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent

Related reference:

- “DROP CONTACTGROUP command” in *Command Reference*
- “contact_host - Location of contact list configuration parameter” in *Performance Guide*
- “db2AddContactGroup - Add a contact group to whom notification messages can be sent” on page 26
- “db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent” on page 101
- “db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent” on page 103
- “db2UpdateContactGroup - Update the attributes of a contact group” on page 338
- “SQLCA ” on page 526
- “DROP CONTACTGROUP command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent” on page 77

db2Export - Export data from a database

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. Label-based access control (LBAC) is enforced for this function. The data that is exported may be limited by the LBAC credentials of the caller if the data is protected by LBAC.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Export (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ExportStruct
{
    char *piDataFileName;
    struct sqlu_media_list *piLobPathList;
    struct sqlu_media_list *piLobFileList;
    struct sqldcol *piDataDescriptor;
    struct sqllob *piActionString;
    char *piFileType;
    struct sqlchar *piFileTypeMod;
    char *piMsgFileName;
    db2int16 iCallerAction;
    struct db2ExportOut *poExportInfoOut;
    struct db2ExportIn *piExportInfoIn;
    struct sqlu_media_list *piXmlPathList;
    struct sqlu_media_list *piXmlFileList;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportIn
{
    db2UInt16 *piXmlSaveSchema;
} db2ExportIn;

typedef SQL_STRUCTURE db2ExportOut
{
    db2UInt64 oRowsExported;
} db2ExportOut;

SQL_API_RC SQL_API_FN
db2gExport (
```

db2Export - Export data from a database

```
    db2UInt32 versionNumber,  
    void * pParmStruct,  
    struct sqlca * pSqlca);  
  
typedef SQL_STRUCTURE db2gExportStruct  
{  
    char *piDataFileName;  
    struct sqlu_media_list *piLobPathList;  
    struct sqlu_media_list *piLobFileList;  
    struct sqldcol *piDataDescriptor;  
    struct sqllob *piActionString;  
    char *piFileType;  
    struct sqlchar *piFileTypeMod;  
    char *piMsgFileName;  
    db2int16 iCallerAction;  
    struct db2ExportOut *poExportInfoOut;  
    db2UInt16 iDataFileNameLen;  
    db2UInt16 iFileTypeLen;  
    db2UInt16 iMsgFileNameLen;  
    struct db2ExportIn *piExportInfoIn;  
    struct sqlu_media_list *piXmlPathList;  
    struct sqlu_media_list *piXmlFileList;  
} db2gExportStruct;
```

db2Export 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 db2ExportStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2ExportStruct data structure parameters:

piDataFileName

Input. A string containing the path and the name of the external file into which the data is to be exported.

piLobPathList

Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files are to be stored. Exported LOB data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piLobFileList

Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its 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 piLobPathList), and then appending a 3-digit sequence number and the .lob extension. For example, if the current LOB path is the directory /u/foo/lob/path, the current LOB file name is bar, and the LOBSINSEPFILLES file type modifier is set, then the created LOB files will be /u/foo/LOB/path/bar.001.lob, /u/foo/LOB/path/bar.002.lob, and so on. If the LOBSINSEPFILLES file

type modifier is not set, then all the LOB documents will be concatenated and put into one file /u/foo/lob/path/bar.001.lob

piDataDescriptor

Input. Pointer to an sqlcol 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 header file, located in the include directory) 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.

piActionString

Input. Pointer to an sqllob structure containing a valid dynamic SQL SELECT statement. The structure contains a 4-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 piDataDescriptor), 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.

piFileType

Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in sqlutil header file) 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.

piFileTypeMod

Input. A pointer to an sqlcol 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. See related link below: "File type modifiers for the export utility."

db2Export - Export data from a database

piMsgFileName

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, the information is appended . If it does not exist, a file is created.

iCallerAction

Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) 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.

poExportInfoOut

A pointer to the db2ExportOut structure.

piExportInfoIn

Input. Pointer to the db2ExportIn structure.

piXmlPathList

Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files are to be stored. Exported XML data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piXmlFileList

Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its 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 XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piXmlFileList), and then appending a 3-digit sequence number and the .xml extension. For example, if the current XML path is the directory /u/foo/xml/path, the current XML file name is bar, and the XMLINSEPFILS file type modifier is set, then the created XML files will be /u/foo/xml/path/bar.001.xml, /u/foo/xml/path/bar.002.xml, and so on. If the XMLINSEPFILS file type

modifier is not set, then all the XML documents will be concatenated and put into one file /u/foo/xml/path/bar.001.xml

db2ExportIn data structure parameters:

piXmlSaveSchema

Input. Indicates that the SQL identifier of the XML schema used to validate each exported XML document should be saved in the exported data file. Possible values are TRUE and FALSE.

db2ExportOut data structure parameters:

oRowsExported

Output. Returns the number of records exported to the target file.

db2gExportStruct data structure specific parameters:

iDataFileNameLen

Input. A 2-byte unsigned integer representing the length in bytes of the data file name.

iFileTypeLen

Input. A 2-byte unsigned integer representing the length in bytes of the file type.

iMsgFileNameLen

Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

Usage notes:

Before starting an export operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

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.

If the export utility produces warnings, the message will be written out to a message file, or standard output if one is not specified.

A warning message is issued if the number of columns (dcolnum field of sqldcol structure) in the external column name array, piDataDescriptor, 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.

DB2 Connect can be used to export tables from DRDA servers such as DB2 for z/OS and OS/390®, DB2 for VM and VSE, and DB2 for iSeries™. Only PC/IXF export is supported.

db2Export - Export data from a database

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.

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 parameter beginning with `SELECT * FROM tablename`, and the piDataDescriptor parameter specifying default names. Indexes are not saved for views, or if the `SELECT` clause of the piActionString includes a join. A `WHERE` clause, a `GROUP BY` clause, or a `HAVING` clause in the piActionString parameter 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. `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 operating system, a single control file, ctrlfile.lst, is used by all Data Links servers. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows 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).

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.

db2Export - Export data from a database

number

A host variable that will contain the number of exported rows.

Related tasks:

- “Exporting data” in *Data Movement Utilities Guide and Reference*

Related reference:

- “sqlchar ” on page 527
- “sqldcol ” on page 530
- “sqllob ” on page 574
- “sqlu_media_list ” on page 582
- “SQLCA ” on page 526
- “ADMIN_CMD procedure – Run administrative commands” in *Administrative SQL Routines and Views*
- “EXPORT command” in *Command Reference*
- “EXPORT command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2Import - Import data into a table, hierarchy, nickname or view” on page 139
- “db2Load - Load data into a table” on page 205

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++)”

File type modifiers for the export utility

Table 6. Valid file type modifiers for the export utility: All file formats

Modifier	Description
lobsinfile	<p><i>lob-path</i> specifies the path to the files containing LOB data.</p> <p>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 <i>filename.ext.nnn.mmm/</i>, where <i>filename.ext</i> is the name of the file that contains the LOB, <i>nnn</i> is the offset in bytes of the LOB within the file, and <i>mmm</i> is the length of the LOB in bytes. For example, if the string <i>db2exp.001.123.456/</i> is stored in the data file, the LOB is located at offset 123 in the file <i>db2exp.001</i>, and is 456 bytes long.</p> <p>If you specify the “<i>lobsinfile</i>” modifier when using EXPORT, the LOB data is placed in the locations specified by the LOBS TO clause. Otherwise the LOB data is sent to the data file directory. The LOBS TO clause specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The LOBS TO or LOBFILE options will implicitly activate the LOBSINFILE behavior.</p> <p>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 <i>db2exp.001.7.-1/</i>.</p>
xmlinsefiles	Each XQuery Data Model (QDM) instance is written to a separate file. By default, multiple values are concatenated together in the same file.
lobsinsefiles	Each LOB value is written to a separate file. By default, multiple values are concatenated together in the same file.
xmlnodeclaration	QDM instances are written without an XML declaration tag. By default, QDM instances are exported with an XML declaration tag at the beginning that includes an encoding attribute.
xmlchar	QDM instances are written in the character codepage. Note that the character codepage is the value specified by the codepage file type modifier, or the application codepage if it is not specified. By default, QDM instances are written out in Unicode.
xmlgraphic	If the <i>xmlgraphic</i> modifier is specified with the EXPORT command, the exported XML document will be encoded in the UTF-16 code page regardless of the application code page or the codepage file type modifier.

Table 7. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format

Modifier	Description
chardelx	<p><i>x</i> 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.² If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:</p> <p style="text-align: center;">modified by <code>chardel"</code></p> <p>The single quotation mark (') can also be specified as a character string delimiter as follows:</p> <p style="text-align: center;">modified by <code>chardel'</code></p>

Table 7. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description
timestampformat="x"	<p>x is the format of the time stamp in the source file.⁴ Valid time stamp elements are:</p> <ul style="list-style-type: none"> YYYY - Year (four digits ranging from 0000 - 9999) M - Month (one or two digits ranging from 1 - 12) MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM) MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM) 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; mutually exclusive with all other microsecond elements) UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microsecond elements) UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microsecond elements) UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microsecond elements) UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microsecond elements) U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microsecond elements) TT - Meridian indicator (AM or PM) <p>Following is an example of a time stamp format: "YYYY/MM/DD HH:MM:SS.UUUUUU"</p> <p>The MMM element will produce the following values: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'. 'Jan' is equal to month 1, and 'Dec' is equal to month 12.</p> <p>The following example illustrates how to export data containing user-defined time stamp formats from a table called 'schedule':</p> <pre>db2 export to delfile2 of del modified by timestampformat="yyyy.mm.dd hh:mm tt" select * from schedule</pre>

File type modifiers for the export utility

Table 8. Valid file type modifiers for the export utility: IXF file format

Modifier	Description
codepage=x	<p>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 from this code page to the application code page during the export operation.</p> <p>For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The codepage modifier cannot be used with the <code>lobsinfile</code> modifier.</p>

Table 9. Valid file type modifiers for the export utility: WSF file format

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a. ⁵ This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0. ⁵
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1. ⁵
4	Creates a WSF file containing DBCS characters.

Notes:

- 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.
- Delimiter restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.
- 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.

- 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)

- 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 can be specified.

6. The WSF file format is not supported for XML columns.
7. All QDM instances are written to XML files that are separate from the main data file, even if neither the "XMLFILE" nor the "XML TO" clause is specified. By default, XML files are written to the path of the exported data file. The default base name for XML files is the name of the exported data file with the ".xml" appended to it.
8. All QDM instances are written with an XML declaration at the beginning that includes an encoding attribute, unless the XMLNODEDECLARATION file type modifier is specified.
9. By default, all QDM instances are written in Unicode unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. The default path for XML data and LOB data is the path of the main data file. The default XML file base name is the main data file. The default LOB file base name is the main data file. For example, if the main data file is
/mypath/myfile.del

, the default path for XML data and LOB data is
/mypath"

, the default XML file base name is
myfile.del

, and the default LOB file base name is
myfile.del

.

The LOBSINFILE file type modifier must be specified in order to have LOB files generated.

11. The export utility appends a numeric identifier to each LOB file or XML file. The identifier is a 3 digit, 0 padded sequence value, starting at
.001

. After the 999th LOB file or XML file, the identifier will no longer be padded with zeroes (for example, the 1000th LOG file or XML file will have an extension of
.1000

. Following the numeric identifier is a three character type identifier representing the data type, either
.lob

or
.xml

. For example, a generated LOB file would have a name in the format
myfile.del.001.lob

, and a generated XML file would be have a name in the format
myfile.del.001.xml

.

12. It is possible to have the export utility export QDM instances that are not well-formed documents by specifying an XQuery. However, you will not be

File type modifiers for the export utility

able to import or load these exported documents directly into an XML column, since XML columns can only contain complete documents.

Related reference:

- “Delimiter restrictions for moving data” on page 237
- “db2Export - Export data from a database” on page 81
- “EXPORT command” in *Command Reference*

db2GetAlertCfg - Get the alert configuration settings for the health indicators

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

API and data structure syntax:

```
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;
    struct db2GetAlertCfgInd *pioIndicators;
} db2GetAlertCfgData;

typedef SQL_STRUCTURE db2GetAlertCfgInd
{
    db2UInt32 ioIndicatorID;
    sqlint64 oAlarm;
    sqlint64 oWarning;
    db2UInt32 oSensitivity;
    char *poFormula;
    db2UInt32 oActionEnabled;
    db2UInt32 oCheckThresholds;
    db2UInt32 oNumTaskActions;
    struct db2AlertTaskAction *poTaskActions;
    db2UInt32 oNumScriptActions;
    struct db2AlertScriptAction *poScriptActions;
    db2UInt32 oDefault;
} db2GetAlertCfgInd;

typedef SQL_STRUCTURE db2AlertTaskAction
{
    char *pTaskName;
    db2UInt32 condition;
    char *pUserID;
    char *pPassword;
    char *pHostName;
} db2AlertTaskAction;

typedef SQL_STRUCTURE db2AlertScriptAction
{
    db2UInt32 scriptType;
```

db2GetAlertCfg - Get the alert configuration settings for the health indicators

```
db2UInt32 condition;  
char *pPathName;  
char *pWorkingDir;  
char *pCmdLineParms;  
char stmtTermChar;  
char *pUserID;  
char *pPassword;  
char *pHostName;  
} db2AlertScriptAction;
```

db2GetAlertCfg 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.

db2GetAlertCfgData data structure parameters:

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.

db2GetAlertCfg - Get the alert configuration settings for the health indicators

pioIndicators

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

db2GetAlertCfgInd data structure parameters:

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.

oDefault

Output. Indicates whether current settings are inherited from the default. Set to TRUE to indicate the current settings are inherited from the default; set to FALSE otherwise.

db2AlertTaskAction data structure parameters:

pTaskname

The name of the task.

condition

The condition for which to run the action.

db2GetAlertCfg - Get the alert configuration settings for the health indicators

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.

db2AlertScriptAction data structure parameters:

scriptType

Specifies the type of 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

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:

db2GetAlertCfg - Get the alert configuration settings for the health indicators

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 call to the `db2GetAlertCfgFree` API whenever the `db2GetAlertCfg` API returns with no error. See `db2ApiDf.h`, located in the include directory, for information about the `db2GetAlertCfgFree` API.

Related tasks:

- “Configuring health indicators using a client application” in *System Monitor Guide and Reference*

Related reference:

- “GET ALERT CONFIGURATION command” in *Command Reference*
- “`db2GetAlertCfgFree` - Free the memory allocated by the `db2GetAlertCfg` API” on page 100
- “`db2ResetAlertCfg` - Reset the alert configuration of health indicators” on page 279
- “`db2UpdateAlertCfg` - Update the alert configuration settings for health indicators” on page 328
- “SQLCA ” on page 526
- “Health indicators” in *System Monitor Guide and Reference*

db2GetAlertCfgFree - Free the memory allocated by the db2GetAlertCfg API

Frees the memory allocated by the db2GetAlertCfg API.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2GetAlertCfgFree (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2GetAlertCfgFree 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.

Related reference:

- “db2GetAlertCfg - Get the alert configuration settings for the health indicators” on page 95
- “db2ResetAlertCfg - Reset the alert configuration of health indicators” on page 279
- “db2UpdateAlertCfg - Update the alert configuration settings for health indicators” on page 328
- “SQLCA ” on page 526

db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent

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`

API and data structure syntax:

```
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;
```

db2GetContactGroup 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 `db2ContactGroupData` structure.

pSqlca

Output. A pointer to the `sqlca` structure.

db2ContactGroupData data structure parameters:

pGroupName

Input. The name of the group to be retrieved.

pDescription

The description of the group.

db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent

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.

db2ContactTypeData data structure parameters:

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.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

Related reference:

- “GET CONTACTGROUP command” in *Command Reference*
- “contact_host - Location of contact list configuration parameter” in *Performance Guide*
- “db2AddContactGroup - Add a contact group to whom notification messages can be sent” on page 26
- “db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent” on page 79
- “db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent” on page 103
- “db2UpdateContactGroup - Update the attributes of a contact group” on page 338
- “SQLCA ” on page 526
- “db2GetContacts - Get the list of contacts to whom notification messages can be sent” on page 105
- “CONTACTGROUPS administrative view – Retrieve the list of contact groups” in *Administrative SQL Routines and Views*

db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent

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.

Authorization:

None

Required connection:

None

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
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;
```

db2GetContactGroups 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.

db2GetContactGroupsData data structure parameters:

ioNumGroups

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

poGroups

Output. A pointer to the `db2ContactGroupDesc` structure.

db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent

db2ContactGroupDesc data structure parameters:

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.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

Related reference:

- "GET CONTACTGROUPS command" in *Command Reference*
- "contact_host - Location of contact list configuration parameter" in *Performance Guide*
- "db2AddContactGroup - Add a contact group to whom notification messages can be sent" on page 26
- "db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent" on page 79
- "db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent" on page 101
- "db2UpdateContactGroup - Update the attributes of a contact group" on page 338
- "SQLCA " on page 526
- "db2GetContacts - Get the list of contacts to whom notification messages can be sent" on page 105
- "CONTACTGROUPS administrative view – Retrieve the list of contact groups" in *Administrative SQL Routines and Views*

db2GetContacts - Get the list of contacts to whom notification messages can be sent

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

API and data structure syntax:

```
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;
```

db2GetContacts 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.

db2GetContactsData data structure parameters:

ioNumContacts

The number of `poContacts`.

poContacts

Output. A pointer to the `db2ContactData` structure. The fields `poContacts`,

db2GetContacts - Get the list of contacts to whom notification messages can be sent

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.

db2ContactData data structure parameters:

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.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

Related reference:

- “GET CONTACTS command” in *Command Reference*
- “db2UpdateContact - Update the attributes of a contact” on page 336
- “contact_host - Location of contact list configuration parameter” in *Performance Guide*
- “db2AddContact - Add a contact to whom notification messages can be sent” on page 24
- “db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent” on page 77
- “SQLCA ” on page 526
- “db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent” on page 101
- “db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent” on page 103
- “CONTACTS administrative view – Retrieve list of contacts” in *Administrative SQL Routines and Views*

db2GetHealthNotificationList - Get the list of contacts to whom health alert notifications can be sent

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

API and data structure syntax:

```
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;
```

db2GetHealthNotificationList 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.

db2GetHealthNotificationListData data structure parameters:

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.

db2GetHealthNotificationList - Get the list of contacts to whom health alert notifications can be sent

db2ContactTypeData data structure parameters:

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:

- "GET HEALTH NOTIFICATION CONTACT LIST command" in *Command Reference*
- "db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent" on page 340
- "SQLCA " on page 526

db2GetRecommendations - Get recommendations to resolve a health indicator in alert state

Retrieves a set of recommendations to resolve a health indicator in alert state on a particular object. The recommendations are returned as an XML document.

Authorization:

None

Required connection:

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

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2GetRecommendations (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2GetRecommendationsData
{
    db2UInt32 iSchemaVersion;
    db2UInt32 iNodeNumber;
    db2UInt32 iIndicatorID;
    db2UInt32 iObjType;
    char *piObjName;
    char *piDbName;
    char *poRecommendation;
} db2GetRecommendationsData;
```

db2GetRecommendations 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 db2GetRecommendationsData structure.

pSqlca

Output. A pointer to the sqlca structure.

db2GetRecommendationsData data structure parameters:

iSchemaVersion

Input. Version ID of the schema used to represent the XML document. The recommendation document will only contain elements or attributes that were defined for that schema version. Set this parameter to: DB2HEALTH_RECSCHEMA_VERSION8_2

iNodeNumber

Input. Specifies the database partition number where the health indicator (HI) entered an alert state. Use the constant SQLM_ALL_NODES to retrieve recommendations for a given object on a given HI across all

db2GetRecommendations - Get recommendations to resolve a health indicator in alert state

database partitions. If the HI has the same recommendations on different database partitions, those recommendations will be grouped into a single recommendation set, where the problem is the group of HIs on different database partitions and the recommendations apply to all of these HIs. To retrieve recommendations on the current database partition, use the constant value `SQLM_CURRENT_NODE`. For standalone instances, `SQLM_CURRENT_NODE` should be used.

iIndicatorID

Input. The health indicator that has entered an alert state and for which we are asking a recommendation. Values are externalized in the header file `sqlmon.h` in the include directory.

iObjType

Input. Specifies the type of object on which the health indicator (identified by `iIndicatorID`) entered an alert state. Value values are:

- `DB2HEALTH_OBJTYPE_DBM`
- `DB2HEALTH_OBJTYPE_DATABASE`
- `DB2HEALTH_OBJTYPE_TABLESPACE`
- `DB2HEALTH_OBJTYPE_TS_CONTAINER`

piObjName

Input. The name of the table space or table space container when the object type parameter, `iObjType`, is set to `DB2HEALTH_OBJTYPE_TABLESPACE` or `DB2HEALTH_OBJTYPE_TS_CONTAINER`. Specify `NULL` if not required. In the case of a table space container, the object name is specified as `<tablespace name>.<container name>`.

piDbname

Input. The alias name for the database on which the HI entered an alert state when the object type parameter, `iObjType`, is `DB2HEALTH_OBJTYPE_TS_CONTAINER`, `DB2HEALTH_OBJTYPE_TABLESPACE`, or `DB2HEALTH_OBJTYPE_DATABASE`. Specify `NULL` otherwise.

poRecommendation

Output. Character pointer that will be set to the address of a buffer in memory containing the recommendation text, formatted as an XML document according to the schema provided in `sqllib/misc/DB2RecommendationSchema.xsd`. The XML document will be encoded in UTF-8, and text in the document will be in the caller's locale.

The `xml:lang` attribute on the `DB2_HEALTH` node will be set to the appropriate client language. The API should be considered as a trusted source and the XML document should not be validated. XML is used as a means of structuring the output data. All memory under this pointer is allocated by the engine and must be freed with a `db2GetRecommendationsFree` call whenever `db2GetRecommendations` returns with no error.

Usage notes:

- Invoke this API to retrieve a set of recommendations to resolve a health alert on a specific DB2 object. If the input health indicator is not in an alert state on the object identified, an error will be returned.
- The recommendations are returned as an XML document, and contain information about actions and scripts that can be run to resolve the alert. Any scripts returned by the API must be executed on the instance on which the

db2GetRecommendations - Get recommendations to resolve a health indicator in alert state

health indicator entered the alert state. For information about the structure and content of the recommendation XML document returned, refer to the schema at `sqllib/misc/DB2RecommendationSchema.xsd`

- All memory allocated by the engine and returned by this function (the recommendation document) must be freed with a `db2GetRecommendationsFree` call whenever `db2GetRecommendations` returns with no error.

Related tasks:

- “Retrieving health recommendations using a client application” in *System Monitor Guide and Reference*

Related reference:

- “GET RECOMMENDATIONS FOR HEALTH INDICATOR command” in *Command Reference*
- “HEALTH_HI_REC procedure” in *Administrative SQL Routines and Views*
- “db2GetRecommendationsFree - Free the memory allocated by the db2GetRecommendations API” on page 112
- “SQLCA ” on page 526

db2GetRecommendationsFree - Free the memory allocated by the db2GetRecommendations API

Frees the memory allocated by the db2GetRecommendations API.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2GetRecommendationsFree (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2GetRecommendationsFree 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 db2GetRecommendationsData structure.

pSqlca

Output. A pointer to the sqlca structure.

Related reference:

- “db2GetRecommendations - Get recommendations to resolve a health indicator in alert state” on page 109
- “SQLCA ” on page 526

db2GetSnapshot - Get a snapshot of the database manager operational status

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
- sysmon

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2GetSnapshot (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2GetSnapshotData
{
    void *piSqlmaData;
    struct sqlm_collected *poCollectedData;
    void *poBuffer;
    db2UInt32 iVersion;
    db2UInt32 iBufferSize;
    db2UInt32 iStoreResult;
    db2int32 iNodeNumber;
    db2UInt32 *poOutputFormat;
    db2UInt32 iSnapshotClass;
} db2GetSnapshotData;

SQL_API_RC SQL_API_FN
db2gGetSnapshot (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gGetSnapshotData
```

db2GetSnapshot - Get a snapshot of the database manager operational status

```
{
    void *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 as the second parameter pParmStruct. To use the structure as described above, specify db2Version810. If you want to use a different version of this structure, check the db2ApiDf.h header file in the include directory for the complete list of supported versions. Ensure that you use the version of the db2GetSnapshotData structure that corresponds to the version number that you specify.

pParmStruct

Input/Output. A pointer to the db2GetSnapshotData structure.

pSqlca

Output. A pointer to the sqlca structure.

db2GetSnapshotData data structure parameters:

piSqlmaData

Input. Pointer to the user-allocated sqlma (monitor area) structure or request data structure, "poRequestData" constructed and returned by the db2AddSnapshotRequest API. The structure specifies the type or types of snapshot data to be collected. If a pointer to the sqlma structure is used, the version passed to the db2GetSnapshot API in the versionNumber parameter should be less than db2Version900 (for example, db2Version810, db2Version822). If a pointer to the request data structure returned by the db2AddSnapshotRequest API in poRequestData parameter is used then the value db2Version900 should be passed in the versionNumber parameter of the db2GetSnapshot API.

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 constants:

db2GetSnapshot - Get a snapshot of the database manager operational status

- 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 the db2GetSnapshotSize API 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 constant value 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. Only allowed when the iVersion parameter is set to SQLM_DBMON_VERSION7 or SQLM_DBMON_VERSION8.
- node value

Note: For standalone instances the SQLM_CURRENT_NODE value 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 header file, located in the include directory) 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.

db2GetSnapshot - Get a snapshot of the database manager operational status

To retrieve a health snapshot with full collection information, use the AGENT_ID field in the SQLMA data structure.

Related tasks:

- “Capturing a database health snapshot from a client application” in *System Monitor Guide and Reference*
- “Capturing a database snapshot from a client application” in *System Monitor Guide and Reference*

Related reference:

- “GET SNAPSHOT command” in *Command Reference*
- “db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API” on page 117
- “db2MonitorSwitches - Get or update the monitor switch settings” on page 246
- “db2AddSnapshotRequest - Add a snapshot request” on page 28
- “db2ConvMonStream - Convert the monitor stream to the pre-version 6 format” on page 57
- “db2ResetMonitor - Reset the database system monitor data” on page 281
- “sqlma ” on page 575
- “SQLCA ” on page 526

Related samples:

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

db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API

Estimates the buffer size needed by the db2GetSnapshot API.

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
- sysmon

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2GetSnapshotSize (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2GetSnapshotSizeData
{
    void *piSqlmaData;
    sqluint32 *poBufferSize;
    db2UInt32 iVersion;
    db2int32 iNodeNumber;
    db2UInt32 iSnapshotClass;
} db2GetSnapshotSizeData;
```

```
SQL_API_RC SQL_API_FN
db2gGetSnapshotSize (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2gGetSnapshotSizeData
{
    void *piSqlmaData;
    sqluint32 *poBufferSize;
```

db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API

```
db2UInt32 iVersion;  
db2int32 iNodeNumber;  
db2UInt32 iSnapshotClass;  
} db2gGetSnapshotSizeData;
```

db2GetSnapshotSize API parameters:

versionNumber

Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct. To use the structure as described above, specify db2Version810. If you want to use a different version of this structure, check the db2ApiDf.h header file in the include directory for the complete list of supported versions. Ensure that you use the version of the db2GetSnapshotSizeStruct structure that corresponds to the version number that you specify.

pParmStruct

Input. A pointer to the db2GetSnapshotSizeStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2GetSnapshotSizeData data structure parameters:

piSqlmaData

Input. Pointer to the user-allocated sqlma (monitor area) structure or request data structure, "poRequestData" constructed and returned by the db2AddSnapshotRequest API. The structure specifies the type or types of snapshot data to be collected. If a pointer to the sqlma structure is used, the version passed to the db2GetSnapshotSize API in the versionNumber parameter should be less than db2Version900 (for example, db2Version810, db2Version822). If a pointer to the request data structure returned by the db2AddSnapshotRequest API in poRequestData parameter is used then the value db2Version900 should be passed in the versionNumber parameter of the db2GetSnapshotSize API.

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. Only allowed when iVersion is set to SQLM_DBMON_VERSION7 or SQLM_DBMON_VERSION8.
- node value

For stand-alone instances, the value, SQLM_CURRENT_NODE must be used.

iSnapshotClass

Input. The class qualifier for the snapshot. Valid values (defined in sqlmon header file, located in the include directory) 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 API 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. 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:

- “Event monitor logical data groups and monitor elements” in *System Monitor Guide and Reference*
- “Snapshot monitor logical data groups and monitor elements” in *System Monitor Guide and Reference*
- “db2AddSnapshotRequest - Add a snapshot request” on page 28
- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “db2MonitorSwitches - Get or update the monitor switch settings” on page 246
- “db2ResetMonitor - Reset the database system monitor data” on page 281
- “sqlma ” on page 575
- “SQLCA ” on page 526

db2GetSyncSession - Get a satellite synchronization session identifier

Gets the satellite's current synchronization session identifier.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2GetSyncSession (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

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

db2GetSyncSession 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.

db2GetSyncSessionStruct data structure parameters:

poSyncSessionID

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

Related reference:

- "SQLCA " on page 526
- "db2QuerySatelliteProgress - Get the status of a satellite synchronization session" on page 252
- "db2SyncSatellite - Start satellite synchronization" on page 325
- "db2SetSyncSession - Set satellite synchronization session" on page 318
- "db2SyncSatelliteStop - Pause satellite synchronization" on page 326
- "db2SyncSatelliteTest - Test whether a satellite can be synchronized" on page 327

db2HADRStart - Start high availability disaster recovery (HADR) operations

Starts HADR operations on a database.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. The API establishes a database connection if one does not exist, and closes the database connection when the API completes.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HADRStart (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRStartStruct
{
    char *piDbAlias;
    char *piUserName;
    char *piPassword;
    db2UInt32 iDbRole;
    db2UInt16 iByForce;
} db2HADRStartStruct;

SQL_API_RC SQL_API_FN
db2gHADRStart (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRStartStruct
{
    char *piDbAlias;
    db2UInt32 iAliasLen;
    char *piUserName;
    db2UInt32 iUserNameLen;
    char *piPassword;
    db2UInt32 iPasswordLen;
    db2UInt32 iDbRole;
    db2UInt16 iByForce;
} db2gHADRStartStruct;
```

db2HADRStart API parameters:

versionNumber

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

db2HADRStart - Start high availability disaster recovery (HADR) operations

pParmStruct

Input. A pointer to the db2HADRStartStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2HADRStartStruct data structure parameters:

piDbAlias

Input. A pointer to the database alias.

piUserName

Input. A pointer to the user name under which the command will be executed.

piPassword

Input. A pointer to a string containing the password.

iDbRole

Input. Specifies which HADR database role should be started on the specified database. Valid values are:

DB2HADR_DB_ROLE_PRIMARY

Start HADR operations on the database in the primary role.

DB2HADR_DB_ROLE_STANDBY

Start HADR operations on the database in the standby role.

iByForce

Input. This argument is ignored if the iDbRole parameter is set to DB2HADR_DB_ROLE_STANDBY. Valid values are:

DB2HADR_NO_FORCE

Specifies that HADR is started on the primary database only if a standby database connects to it within a prescribed time limit.

DB2HADR_FORCE

Specifies that HADR is to be started by force, without waiting for the standby database to connect to the primary database.

db2gHADRStartStruct data structure specific parameters:

iAliasLen

Input. Specifies the length in bytes of the database alias.

iUserNameLen

Input. Specifies the length in bytes of the user name.

iPasswordLen

Input. Specifies the length in bytes of the password.

Related tasks:

- “Initializing high availability disaster recovery (HADR)” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “SQLCA ” on page 526
- “db2HADRStop - Stop high availability disaster recovery (HADR) operations” on page 123
- “db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database” on page 125

db2HADRStop - Stop high availability disaster recovery (HADR) operations

Stops HADR operations on a database.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. The API establishes a database connection if one does not exist, and closes the database connection when the API completes.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HADRStop (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRStopStruct
{
    char *piDbAlias;
    char *piUserName;
    char *piPassword;
} db2HADRStopStruct;

SQL_API_RC SQL_API_FN
db2gHADRStop (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRStopStruct
{
    char *piDbAlias;
    db2UInt32 iAliasLen;
    char *piUserName;
    db2UInt32 iUserNameLen;
    char *piPassword;
    db2UInt32 iPasswordLen;
} db2gHADRStopStruct;
```

db2HADRStop 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 db2HADRStopStruct structure.

db2HADRStop - Stop high availability disaster recovery (HADR) operations

pSqlca

Output. A pointer to the sqlca structure.

db2HADRStopStruct data structure parameters:

piDbAlias

Input. A pointer to the database alias.

piUserName

Input. A pointer to the user name under which the command will be executed.

piPassword

Input. A pointer to a string containing the password.

db2gHADRStopStruct data structure specific parameters:

iAliasLen

Input. Specifies the length in bytes of the database alias.

iUserNameLen

Input. Specifies the length in bytes of the user name.

iPasswordLen

Input. Specifies the length in bytes of the password.

Related tasks:

- “Stopping high availability disaster recovery (HADR)” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “SQLCA ” on page 526
- “db2HADRStart - Start high availability disaster recovery (HADR) operations” on page 121
- “db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database” on page 125

db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database

Instructs a standby database to take over as the primary database. This API can be called against a standby database only.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. The API establishes a database connection if one does not exist, and closes the database connection when the API completes.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HADRTakeover (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HADRTakeoverStruct
{
    char *piDbAlias;
    char *piUserName;
    char *piPassword;
    db2UInt16 iByForce;
} db2HADRTakeoverStruct;

SQL_API_RC SQL_API_FN
db2gHADRTakeover (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHADRTakeoverStruct
{
    char *piDbAlias;
    db2UInt32 iAliasLen;
    char *piUserName;
    db2UInt32 iUserNameLen;
    char *piPassword;
    db2UInt32 iPasswordLen;
    db2UInt16 iByForce;
} db2gHADRTakeoverStruct;
```

db2HADRTakeover API parameters:

versionNumber

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

db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database

pParmStruct

Input. A pointer to the db2HADRTakeoverStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2HADRTakeoverStruct data structure parameters:

piDbAlias

Input. A pointer to the database alias.

piUserName

Input. A pointer to the user name under which the command will be executed.

piPassword

Input. A pointer to a string containing the password.

iByForce

Input. Valid values are:

DB2HADR_NO_FORCE

Specifies that a takeover occurs only if the two systems are in peer state with communication established; this results in a role reversal between the HADR primary and HADR standby databases.

DB2HADR_FORCE

Specifies that the standby database takes over as the primary database without waiting for confirmation that the original primary database has been shut down. Forced takeover must be issued when the standby database is in either remote catchup pending or peer state.

db2gHADRTakeoverStruct data structure specific parameters:

iAliasLen

Input. Specifies the length in bytes of the database alias.

iUserNameLen

Input. Specifies the length in bytes of the user name.

iPasswordLen

Input. Specifies the length in bytes of the password.

Related tasks:

- “Switching database roles in high availability disaster recovery (HADR)” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “SQLCA ” on page 526
- “db2HADRTakeover - Start high availability disaster recovery (HADR) operations” on page 121
- “db2HADRTakeover - Stop high availability disaster recovery (HADR) operations” on page 123

db2HistoryCloseScan - End the 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 the db2HistoryOpenScan API.

Authorization:

None

Required connection:

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

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HistoryCloseScan (
    db2Uint32 versionNumber,
    void * piHandle,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
db2gHistoryCloseScan (
    db2Uint32 versionNumber,
    void * piHandle,
    struct sqlca * pSqlca);
```

db2HistoryCloseScan API parameters:

versionNumber

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 the db2HistoryOpenScan API.

pSqlca

Output. A pointer to the sqlca structure.

Usage notes:

For a detailed description of the use of the history file APIs, refer to the db2HistoryOpenScan API.

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.

Related reference:

- “LIST HISTORY command” in *Command Reference*
- “db2HistoryGetEntry - Get the next entry in the history file” on page 129

db2HistoryCloseScan - End the history file scan

- “db2HistoryOpenScan - Start a history file scan” on page 132
- “db2HistoryUpdate - Update a history file entry” on page 136
- “db2Prune - Delete the history file entries or log files from the active log path” on page 249
- “SQLCA ” on page 526

Related samples:

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

db2HistoryGetEntry - Get the next entry in the history file

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

Authorization:

None

Required connection:

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

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HistoryGetEntry (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryGetEntryStruct
{
    struct db2HistoryData *pioHistData;
    db2UInt16 iHandle;
    db2UInt16 iCallerAction;
} db2HistoryGetEntryStruct;

SQL_API_RC SQL_API_FN
db2gHistoryGetEntry (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2HistoryGetEntry 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 db2HistoryGetEntryStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2HistoryGetEntryStruct data structure parameters:

pioHistData

Input. A pointer to the db2HistData structure.

iHandle

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

iCallerAction

Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2HistoryGetEntry - Get the next entry in the history file

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.

Usage notes:

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

For a detailed description of the use of the history file APIs, refer to the db2HistoryOpenScan API.

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

db2HistoryGetEntry - Get the next entry in the history file

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).

Related reference:

- "LIST HISTORY command" in *Command Reference*
- "db2HistoryData " on page 507
- "db2HistoryCloseScan - End the history file scan" on page 127
- "db2HistoryOpenScan - Start a history file scan" on page 132
- "db2HistoryUpdate - Update a history file entry" on page 136
- "db2Prune - Delete the history file entries or log files from the active log path" on page 249
- "SQLCA " on page 526
- "DB_HISTORY administrative view – Retrieve history file information" in *Administrative SQL Routines and Views*

Related samples:

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

db2HistoryOpenScan - Start a history file scan

This API starts a history file scan.

Authorization:

None

Required connection:

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

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HistoryOpenScan (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryOpenStruct
{
    char *piDatabaseAlias;
    char *piTimestamp;
    char *piObjectName;
    db2UInt32 oNumRows;
    db2UInt32 oMaxTbspaces;
    db2UInt16 iCallerAction;
    db2UInt16 oHandle;
} db2HistoryOpenStruct;

SQL_API_RC SQL_API_FN
db2gHistoryOpenScan (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHistoryOpenStruct
{
    char *piDatabaseAlias;
    char *piTimestamp;
    char *piObjectName;
    db2UInt32 iAliasLen;
    db2UInt32 iTimestampLen;
    db2UInt32 iObjectNameLen;
    db2UInt32 oNumRows;
    db2UInt32 oMaxTbspaces;
    db2UInt16 iCallerAction;
    db2UInt16 oHandle;
} db2gHistoryOpenStruct;
```

db2HistoryOpenScan API parameters:

versionNumber

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

pParmStruct

Input or Output. A pointer to the `db2HistoryOpenStruct` data structure.

pSqlca

Output. A pointer to the sqlca structure.

db2HistoryOpenStruct data structure parameters:

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 call, this parameter contains the number of matching history file entries.

oMaxTbspaces

Output. The maximum number of table space names stored with any history entry.

iCallerAction

Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

db2HistoryOpenScan - Start a history file scan

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 the db2HistoryGetEntry, and db2HistoryCloseScan APIs.

db2gHistoryOpenStruct data structure specific parameters:

iAliasLen

Input. Specifies the length in bytes of the database alias string.

iTimestampLen

Input. Specifies the length in bytes of the timestamp string.

iObjectNameLen

Input. Specifies the length in bytes of the object name string.

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 the db2HistoryOpenScan API, which returns parameter value oNumRows.
2. Allocate a db2HistData structure with space for n oTablespace fields, where n is an arbitrary number.
3. Set the iNumTablespaces field of the db2HistoryData structure to n.
4. In a loop, perform the following:
 - Call the db2HistoryGetEntry API to fetch from the history file.
 - If db2HistoryGetEntry API returns an SQLCODE value of SQL_RC_OK, use the oNumTablespaces field of the db2HistoryData structure to determine the number of table space entries returned.
 - If db2HistoryGetEntry API returns an SQLCODE value 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 db2HistoryData structure with enough space for oDB2UsedTablespace table space entries, and set iDB2NumTablespace to oDB2UsedTablespace.
 - If db2HistoryGetEntry API returns an SQLCODE value of SQLE_RC_NOMORE, all history file entries have been retrieved.

db2HistoryOpenScan - Start a history file scan

- Any other SQLCODE indicates a problem.
5. When all of the information has been fetched, call the db2HistoryCloseScan API to free the resources allocated by the call to db2HistoryOpenScan.

The macro SQLUHINFOSIZE(n) (defined in sqlutil header file) is provided to help determine how much memory is required for a db2HistoryData structure with space for n oTablespace entries.

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 that is to have its history file 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.

Related reference:

- “LIST HISTORY command” in *Command Reference*
- “db2HistoryCloseScan - End the history file scan” on page 127
- “db2HistoryGetEntry - Get the next entry in the history file” on page 129
- “db2HistoryUpdate - Update a history file entry” on page 136
- “db2Prune - Delete the history file entries or log files from the active log path” on page 249
- “SQLCA ” on page 526

Related samples:

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

db2HistoryUpdate - Update a history file entry

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2HistoryUpdate (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2HistoryUpdateStruct
{
    char *piNewLocation;
    char *piNewDeviceType;
    char *piNewComment;
    char *piNewStatus;
    db2HistoryEID iEID;
} db2HistoryUpdateStruct;

typedef SQL_STRUCTURE db2HistoryEID
{
    SQL_PDB_NODE_TYPE ioNode;
    db2UInt32 ioHID;
} db2HistoryEID;

SQL_API_RC SQL_API_FN
db2gHistoryUpdate (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gHistoryUpdateStruct
{
    char *piNewLocation;
    char *piNewDeviceType;
    char *piNewComment;
    char *piNewStatus;
    db2UInt32 iNewLocationLen;
    db2UInt32 iNewDeviceLen;
    db2UInt32 iNewCommentLen;
    db2UInt32 iNewStatusLen;
    db2HistoryEID iEID;
} db2gHistoryUpdateStruct;
```

db2HistoryUpdate - Update a history file entry

db2HistoryUpdate 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 db2HistoryUpdateStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2HistoryUpdateStruct data structure parameters:

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.

piNewStatus

Input. A pointer to a string specifying a new status type for the entry. Setting this parameter to NULL, or pointing to zero, leaves the status unchanged.

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

db2HistoryEID data structure parameters:

ioNode

This parameter can be used as either an input or output parameter. Indicates the node number.

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

Indicates the local history file entry ID.

db2gHistoryUpdateStruct data structure specific parameters:

iNewLocationLen

Input. Specifies the length in bytes of the piNewLocation parameter.

iNewDeviceLen

Input. Specifies the length in bytes of the piNewDeviceType parameter.

iNewCommentLen

Input. Specifies the length in bytes of the piNewComment parameter.

iNewStatusLen

Input. Specifies the length in bytes of the piNewStatus paramter.

Usage notes:

db2HistoryUpdate - Update a history file entry

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.

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).

Related reference:

- “db2Backup - Back up a database or table space” on page 41
- “db2HistoryCloseScan - End the history file scan” on page 127
- “db2HistoryGetEntry - Get the next entry in the history file” on page 129
- “db2HistoryOpenScan - Start a history file scan” on page 132
- “db2Prune - Delete the history file entries or log files from the active log path” on page 249
- “db2Rollforward - Roll forward a database” on page 296
- “SQLCA ” on page 526
- “UPDATE HISTORY command” in *Command Reference*

Related samples:

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

db2Import - Import data into a table, hierarchy, nickname or view

Inserts data from an external file with a supported file format into a table, hierarchy, nickname or view. The load utility is faster than this function. The load utility, however, does not support loading data at the hierarchy level or loading into a nickname.

Authorization:

- IMPORT using the INSERT option requires one of the following:
 - sysadm
 - dbadm
 - CONTROL privilege on each participating table, view or nickname
 - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
 - sysadm
 - dbadm
 - CONTROL privilege on the table, view or nickname
 - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
 - sysadm
 - dbadm
 - CONTROL privilege on the table or view
 - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:
 - sysadm
 - dbadm
 - CREATETAB authority on the database and USE privilege on the table space, as well as one of:
 - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
 - CREATIN privilege on the schema, if the schema name of the table refers to an existing schema
- 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:

db2Import - Import data into a table, hierarchy, nickname or view

- 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.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Import (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ImportStruct
{
    char *piDataFileName;
    struct sqlu_media_list *piLobPathList;
    struct sqldcol *piDataDescriptor;
    struct sqlchar *piActionString;
    char *piFileType;
    struct sqlchar *piFileTypeMod;
    char *piMsgFileName;
    db2int16 iCallerAction;
    struct db2ImportIn *piImportInfoIn;
    struct db2ImportOut *poImportInfoOut;
    db2int32 *piNullIndicators;
    struct sqlu_media_list *piXmlPathList;
} db2ImportStruct;

typedef SQL_STRUCTURE db2ImportIn
{
    db2UInt64 iRowcount;
    db2UInt64 iRestartcount;
    db2UInt64 iSkipcount;
    db2int32 *piCommitcount;
    db2UInt32 iWarningcount;
    db2UInt16 iNoTimeout;
    db2UInt16 iAccessLevel;
    db2UInt16 *piXmlParse;
    struct db2DMUXmlValidate *piXmlValidate;
} db2ImportIn;

typedef SQL_STRUCTURE db2ImportOut
{
    db2UInt64 oRowsRead;
    db2UInt64 oRowsSkipped;
    db2UInt64 oRowsInserted;
    db2UInt64 oRowsUpdated;
    db2UInt64 oRowsRejected;
    db2UInt64 oRowsCommitted;
} db2ImportOut;

typedef SQL_STRUCTURE db2DMUXmlMapSchema
{
    struct db2Char iMapFromSchema;
    struct db2Char iMapToSchema;
} db2DMUXmlMapSchema;
```

db2Import - Import data into a table, hierarchy, nickname or view

```
typedef SQL_STRUCTURE db2DMUXmlValidateXds
{
    struct db2Char *piDefaultSchema;
    db2UInt32 iNumIgnoreSchemas;
    struct db2Char *piIgnoreSchemas;
    db2UInt32 iNumMapSchemas;
    struct db2DMUXmlMapSchema *piMapSchemas;
} db2DMUXmlValidateXds;

typedef SQL_STRUCTURE db2DMUXmlValidateSchema
{
    struct db2Char *piSchema;
} db2DMUXmlValidateSchema;

typedef SQL_STRUCTURE db2DMUXmlValidate
{
    db2UInt16 iUsing;
    struct db2DMUXmlValidateXds *piXdsArgs;
    struct db2DMUXmlValidateSchema *piSchemaArgs;
} db2DMUXmlValidate;

SQL_API_RC SQL_API_FN
db2gImport (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gImportStruct
{
    char *piDataFileName;
    struct sqlu_media_list *piLobPathList;
    struct sqldcol *piDataDescriptor;
    struct sqlchar *piActionString;
    char *piFileType;
    struct sqlchar *piFileTypeMod;
    char *piMsgFileName;
    db2int16 iCallerAction;
    struct db2gImportIn *piImportInfoIn;
    struct db2gImportOut *piImportInfoOut;
    db2int32 *piNullIndicators;
    db2UInt16 iDataFileNameLen;
    db2UInt16 iFileTypeLen;
    db2UInt16 iMsgFileNameLen;
    struct sqlu_media_list *piXmlPathList;
} db2gImportStruct;

typedef SQL_STRUCTURE db2gImportIn
{
    db2UInt64 iRowcount;
    db2UInt64 iRestartcount;
    db2UInt64 iSkipcount;
    db2int32 *piCommitcount;
    db2UInt32 iWarningcount;
    db2UInt16 iNoTimeout;
    db2UInt16 iAccessLevel;
    db2UInt16 *piXmlParse;
    struct db2DMUXmlValidate *piXmlValidate;
} db2gImportIn;

typedef SQL_STRUCTURE db2gImportOut
{
    db2UInt64 oRowsRead;
    db2UInt64 oRowsSkipped;
    db2UInt64 oRowsInserted;
```

db2Import - Import data into a table, hierarchy, nickname or view

```
    db2UInt64 oRowsUpdated;  
    db2UInt64 oRowsRejected;  
    db2UInt64 oRowsCommitted;  
} db2gImportOut;
```

db2Import 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 db2ImportStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2ImportStruct data structure parameters:

piDataFileName

Input. A string containing the path and the name of the external input file from which the data is to be imported.

piLobPathList

Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files can be found. This parameter is not valid when you import to a nickname.

piDataDescriptor

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 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 piDataDescriptor is NULL, or is set to SQL_METH_D, default selection of columns from the external input file is done. In

db2Import - Import data into a table, hierarchy, nickname or view

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.

piActionString

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]}
[DATA LINK SPECIFICATION data link-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 indexxf is in FileTypeMod, and FileType is SQL_IXF.) If the table is not already defined, an error is returned.

Note: If an error occurs after the existing data is deleted, that data is lost.

This parameter is not valid when you import to a nickname.

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. This parameter is not valid when you import to a nickname.

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.

db2Import - Import data into a table, hierarchy, nickname or view

Note: If an error occurs after the existing data is deleted, that data is lost.

This parameter is not valid when you import to a nickname.

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 Manager. 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

db2Import - Import data into a table, hierarchy, nickname or view

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).

This parameter is not valid when you import to a nickname.

piFileType

Input. A string that indicates the format of the data within the external file. Supported external file formats 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. The WSF file type is not supported when you import to a nickname.

piFileTypeMod

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. See related link "File type modifiers for the import utility".

piMsgFileName

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.

iCallerAction

Input. An action requested by the caller. Valid values 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.

db2Import - Import data into a table, hierarchy, nickname or view

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.

piImportInfoIn

Input. Pointer to the db2ImportIn structure.

poImportInfoOut

Output. Pointer to the db2ImportOut structure.

piNullIndicators

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 piDataDescriptor 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.

piXmlPathList

Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files can be found.

db2ImportIn data structure parameters:

iRowcount

Input. The number of physical records to be loaded. Allows a user to load only the first iRowcount rows in a file. If iRowcount is 0, import will attempt to process all the rows from the file.

iRestartcount

Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iSkipcount parameter. iRestartcount and iSkipcount parameters are mutually exclusive.

iSkipcount

Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iRestartcount.

piCommitcount

Input. The number of records to import before committing them to the database. A commit is performed whenever piCommitcount records are imported. A NULL value specifies the default commit count value, which is zero for offline import and AUTOMATIC for online import. Commitcount AUTOMATIC is specified by passing in the value DB2IMPORT_COMMIT_AUTO.

iWarningcount

Input. Stops the import operation after iWarningcount warnings. Set this parameter if no warnings are expected, but verification that the correct file

db2Import - Import data into a table, hierarchy, nickname or view

and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail.

If `iWarningcount` is 0, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

iNoTimeout

Input. Specifies that the import utility will not time out while waiting for locks. This option supersedes the `locktimeout` database configuration parameter. Other applications are not affected. Valid values are:

DB2IMPORT_LOCKTIMEOUT

Indicates that the value of the `locktimeout` configuration parameter is respected.

DB2IMPORT_NO_LOCKTIMEOUT

Indicates there is no timeout.

iAccessLevel

Input. Specifies the access level. Valid values are:

- SQLU_ALLOW_NO_ACCESS

Specifies that the import utility locks the table exclusively.

- SQLU_ALLOW_WRITE_ACCESS

Specifies that the data in the table should still be accessible to readers and writers while the import is in progress.

An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the `REPLACE`, `CREATE`, or `REPLACE_CREATE` import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the `piCommitCount` parameter was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and `piCommitCount` parameter must be specified with a valid number (`AUTOMATIC` is not considered a valid option).

piXmlParse

Input. Type of parsing that should occur for XML documents. Valid values found in the `db2ApiDf` header file in the include directory, are:

DB2DMU_XMLPARSE_PRESERVE_WS

Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS

Whitespace should be stripped.

piXmlValidate

Input. Pointer to the `db2DMUXmlValidate` structure. Indicates that XML schema validation should occur for XML documents.

db2ImportOut data structure parameters:

oRowsRead

Output. Number of records read from the file during import.

oRowsSkipped

Output. Number of records skipped before inserting or updating begins.

db2Import - Import data into a table, hierarchy, nickname or view

oRowsInserted

Output. Number of rows inserted into the target table.

oRowsUpdated

Output. Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).

oRowsRejected

Output. Number of records that could not be imported.

oRowsCommitted

Output. Number of records imported successfully and committed to the database.

db2DMUXmlMapSchema data structure parameters:

iMapFromSchema

Input. The SQL identifier of the XML schema to map from.

iMapToSchema

Input. The SQL identifier of the XML schema to map to.

db2DMUXmlValidateXds data structure parameters:

piDefaultSchema

Input. The SQL identifier of the XML schema that should be used for validation when an XDS does not contain an SCH attribute.

iNumIgnoreSchemas

Input. The number of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

piIgnoreSchemas

Input. The list of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

iNumMapSchemas

Input. The number of XML schemas that will be mapped during XML schema validation. The first schema in the schema map pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

piMapSchemas

Input. The list of XML schema pairs, where each pair represents a mapping of one schema to a different one. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

db2DMUXmlValidateSchema data structure parameters:

piSchema

Input. The SQL identifier of the XML schema to use.

db2DMUXmlValidate data structure parameters:

iUsing

Input. A specification of what to use to perform XML schema validation. Valid values found in the db2ApiDf header file in the include directory, are:

db2Import - Import data into a table, hierarchy, nickname or view

- DB2DMU_XMLVAL_XDS

Validation should occur according to the XDS. This corresponds to the CLP "XMLVALIDATE USING XDS" clause.

- DB2DMU_XMLVAL_SCHEMA

Validation should occur according to a specified schema. This corresponds to the CLP "XMLVALIDATE USING SCHEMA" clause.

- DB2DMU_XMLVAL_SCHEMALOC_HINTS

Validation should occur according to schemaLocation hints found within the XML document. This corresponds to the "XMLVALIDATE USING SCHEMALOCATION HINTS" clause.

piXdsArgs

Input. Pointer to a db2DMUXmlValidateXds structure, representing arguments that correspond to the CLP "XMLVALIDATE USING XDS" clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_XDS.

piSchemaArgs

Input. Pointer to a db2DMUXmlValidateSchema structure, representing arguments that correspond to the CLP "XMLVALIDATE USING SCHEMA" clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_SCHEMA.

db2gImportStruct data structure specific parameters:

iDataFileNameLen

Input. Specifies the length in bytes of piDataFileName parameter.

iFileTypeLen

Input. Specifies the length in bytes of piFileType parameter.

iMsgFileNameLen

Input. Specifies the length in bytes of piMsgFileName parameter.

Usage notes:

Before starting an import operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

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.

db2Import - Import data into a table, hierarchy, nickname or view

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 iRestartcount 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 *piCommitcount 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 60 KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

db2Import - Import data into a table, hierarchy, nickname or view

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 piDataDescriptor, or specifying an explicit list of table columns in piActionString, 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 operating system:

- Importing logically split PC/IXF files is not supported.

db2Import - Import data into a table, hierarchy, nickname or view

- Importing bad format PC/IXF or WSF files is not supported.

Federated considerations:

When using the db2Import API and the INSERT, UPDATE, or INSERT_UPDATE parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you wish to use when doing an import operation already exists.

Related tasks:

- “Importing data” in *Data Movement Utilities Guide and Reference*

Related reference:

- “IMPORT Command” in *Command Reference*
- “SQLCA ” on page 526
- “sqldcol ” on page 530
- “sqlu_media_list ” on page 582
- “IMPORT command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2Export - Export data from a database” on page 81
- “db2Load - Load data into a table” on page 205

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)”
- “tbmove.sqc -- How to move table data (C)”
- “dtformat.sqc -- Load and import data format extensions (C)”
- “tbmove.sqC -- How to move table data (C++)”

File type modifiers for the import utility

Table 10. Valid file type modifiers for the import utility: All file formats

Modifier	Description
compound= <i>x</i>	<p><i>x</i> is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and <i>x</i> statements will be attempted each time.</p> <p>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.</p> <p>This modifier is incompatible with INSERT_UPDATE mode, hierarchical tables, and the following modifiers: usedefaults, identitymissing, identityignore, generatedmissing, and generatedignore.</p>
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	<p><i>lob-path</i> specifies the path to the files containing LOB data.</p> <p>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 <i>filename.ext.nnn.mmm/</i>, where <i>filename.ext</i> is the name of the file that contains the LOB, <i>nnn</i> is the offset in bytes of the LOB within the file, and <i>mmm</i> 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.</p> <p>The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the IMPORT utility the list of paths to search for the LOB files while importing the data.</p> <p>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/.</p>

File type modifiers for the import utility

Table 10. Valid file type modifiers for the import utility: All file formats (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	<p>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:</p> <ul style="list-style-type: none"> • 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.
norowwarnings	Suppresses all warnings about rejected rows.
seclabelchar	<p>Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. IMPORT converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.</p> <p>This modifier cannot be specified if the seclabelname modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned.</p>
seclabelname	<p>Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. IMPORT will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.</p> <p>This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the import fails and an error (SQLCODE SQL3525N) is returned. Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.</p>
usedefaults	<p>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:</p> <ul style="list-style-type: none"> • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (" , ") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. <p>Note: For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL.</p> <p>Without this option, if a source column contains no data for a row instance, one of the following occurs:</p> <ul style="list-style-type: none"> • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 11. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage= <i>x</i>	<p><i>x</i> is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the import operation.</p> <p>The following rules apply:</p> <ul style="list-style-type: none"> • 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 and x7F, inclusive. This refers to ASCII symbols and code points. <p>Notes:</p> <ol style="list-style-type: none"> 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 might be truncated and loss of data can occur.
dateformat=" <i>x</i> "	<p><i>x</i> is the format of the date in the source file.² Valid date elements are:</p> <p>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)</p> <p>A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:</p> <p>"D-M-YYYY" "MM.DD.YYYY" "YYYYDDD"</p>
implieddecimal	<p>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, <i>not</i> 12345.00.</p>

File type modifiers for the import utility

Table 11. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
timeformat="x"	<p>x is the format of the time in the source file.² Valid time elements are:</p> <ul style="list-style-type: none"> 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) <p>A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:</p> <pre> "HH:MM:SS" "HH.MM TT" "SSSSS" </pre>

Table 11. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
timestampformat="x"	<p>x is the format of the time stamp in the source file.² Valid time stamp elements are:</p> <ul style="list-style-type: none"> YYYY - Year (four digits ranging from 0000 - 9999) M - Month (one or two digits ranging from 1 - 12) MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM) MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM) 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; mutually exclusive with all other microsecond elements) UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microsecond elements) UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microsecond elements) UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microsecond elements) UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microsecond elements) U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microsecond elements) TT - Meridian indicator (AM or PM) <p>A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:</p> <pre style="margin-left: 40px;">"YYYY/MM/DD HH:MM:SS.UUUUUU"</pre> <p>The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.</p> <p>The following example illustrates how to import data containing user defined date and time formats into a table called schedule:</p> <pre style="margin-left: 40px;">db2 import from delfile2 of del modified by timestampformat="yyyy.mm.dd hh:mm tt" insert into schedule</pre>

File type modifiers for the import utility

Table 11. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	<p>If usegraphiccodepage is given, the assumption is made that data being imported into graphic or double-byte character large object (DBCLOB) data fields is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic code page is associated with the character code page. IMPORT determines the character code page through either the codepage modifier, if it is specified, or through the code page of the application if the codepage modifier is not specified.</p> <p>This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data.</p> <p>Restrictions</p> <p>The usegraphiocodepage modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The usegraphiocodepage modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.</p>
xmlchar	<p>Specifies that XML documents are encoded in the character code page.</p> <p>This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration.</p> <p>For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the codepage file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.</p>
xmlgraphic	<p>Specifies that XML documents are encoded in the specified graphic code page.</p> <p>This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration.</p> <p>For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the codepage file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute.</p> <p>Note: If the xmlgraphic modifier is specified with the IMPORT command, the XML document to be imported must be encoded in the UTF-16 code page. Otherwise, the XML document may be rejected with a parsing error, or it may be imported into the table with data corruption.</p>

Table 12. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format

Modifier	Description
nochecklengths	<p>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.</p>

Table 12. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format (continued)

Modifier	Description
nullindchar= <i>x</i>	<p><i>x</i> is a single character. Changes the character denoting a null value to <i>x</i>. The default value of <i>x</i> is Y.³</p> <p>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.</p>
reclen= <i>x</i>	<p><i>x</i> is an integer with a maximum value of 32 767. <i>x</i> characters are read for each row, and a new-line character is not used to indicate the end of the row.</p>
striptblanks	<p>Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.</p> <p>In the following example, <code>striptblanks</code> causes the import utility to truncate trailing blank spaces:</p> <pre>db2 import from myfile.asc of asc modified by striptblanks method 1 (1 10, 12 15) messages msgs.txt insert into staff</pre> <p>This option cannot be specified together with <code>striptnulls</code>. These are mutually exclusive options. This option replaces the obsolete <code>t</code> option, which is supported for back-level compatibility only.</p>
striptnulls	<p>Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept.</p> <p>This option cannot be specified together with <code>striptblanks</code>. These are mutually exclusive options. This option replaces the obsolete <code>padwithzero</code> option, which is supported for back-level compatibility only.</p>

Table 13. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format

Modifier	Description
chardel <i>x</i>	<p><i>x</i> 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.³⁴ If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:</p> <pre>modified by chardel'"</pre> <p>The single quotation mark (') can also be specified as a character string delimiter. In the following example, <code>chardel''</code> causes the import utility to interpret any single quotation mark (') it encounters as a character string delimiter:</p> <pre>db2 "import from myfile.del of del modified by chardel'' method p (1, 4) insert into staff (id, years)"</pre>
coldel <i>x</i>	<p><i>x</i> 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.³⁴</p> <p>In the following example, <code>coldel;</code> causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:</p> <pre>db2 import from myfile.del of del modified by coldel; messages msgs.txt insert into staff</pre>
decplusblank	<p>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.</p>

File type modifiers for the import utility

Table 13. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format (continued)

Modifier	Description
decptx	<p>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.³⁴</p> <p>In the following example, decpt; causes the import utility to interpret any semicolon (;) it encounters as a decimal point:</p> <pre>db2 "import from myfile.del of del modified by chardel' decpt; messages msgs.txt insert into staff"</pre>
delprioritychar	<p>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:</p> <pre>db2 import ... modified by delprioritychar ...</pre> <p>For example, given the following DEL data file:</p> <pre>"Smith, Joshua",4000,34.98<row delimiter> "Vincent,<row delimiter>, is a manager", 4005,44.37<row delimiter></pre> <p>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 <i>not</i> specified, there will be three rows in this data file, each delimited by a <row delimiter>.</p>
keepblanks	<p>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.</p>
nochardel	<p>The import utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption.</p> <p>This option cannot be specified with chardelx, delprioritychar or nodoubledel. These are mutually exclusive options.</p>
nodoubledel	<p>Suppresses recognition of double character delimiters.</p>

Table 14. Valid file type modifiers for the import utility: IXF file format

Modifier	Description
forcein	<p>Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages.</p> <p>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.</p>
indexif	<p>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 <i>insert-column</i> is specified.</p>

Table 14. Valid file type modifiers for the import utility: IXF file format (continued)

Modifier	Description
<code>indexschema=schema</code>	Uses the specified <i>schema</i> for the index name during index creation. If <i>schema</i> is not specified (but the keyword <code>indexschema</code> is specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
<code>nochecklengths</code>	If <code>nochecklengths</code> 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.
<code>forcecreate</code>	Specifies that the table should be created with possible missing or limited information after returning SQL3311N during an import operation.

Table 15. IMPORT behavior when using `codepage` and `usegraphiccodepage`

<code>codepage=N</code>	<code>usegraphiccodepage</code>	IMPORT behavior
Absent	Absent	All data in the file is assumed to be in the application code page.
Present	Absent	All data in the file is assumed to be in code page N. Warning: Graphic data will be corrupted when imported into the database if N is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the application code page. Graphic data is assumed to be in the code page of the application graphic data. If the application code page is single-byte, then all data is assumed to be in the application code page. Warning: If the application code page is single-byte, graphic data will be corrupted when imported into the database, even if the database contains graphic columns.
Present	Present	Character data is assumed to be in code page N. Graphic data is assumed to be in the graphic code page of N. If N is a single-byte or double-byte code page, then all data is assumed to be in code page N. Warning: Graphic data will be corrupted when imported into the database if N is a single-byte code page.

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. 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.

File type modifiers for the import utility

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)
```

Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).

3. 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. Delimiter restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.
5. The following file type modifiers are not allowed when importing into a nickname:
 - indexixf
 - indexschema
 - dldelfiletype
 - nodefaults
 - usedefaults
 - no_type_idfiletype
 - generatedignore
 - generatedmissing
 - identityignore
 - identitymissing
 - lobsinfile
6. The WSF file format is not supported for XML columns.
7. The CREATE mode is not supported for XML columns.
8. All XML data must reside in XML files that are separate from the main data file. An XML Data Specifier (XDS) (or a NULL value) must exist for each XML column in the main data file.
9. XML documents are assumed to be in Unicode format or to contain a declaration tag that includes an encoding attribute, unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. Rows containing documents that are not well-formed will be rejected.

11. If the XMLVALIDATE option is specified, documents that successfully validate against their matching schema will be annotated with the schema information as they are inserted. Rows containing documents that fail to validate against their matching schema will be rejected. To successfully perform the validation, the privileges held by the user invoking the import must include at least one of the following:
 - SYSADM or DBADM authority
 - USAGE privilege on the XML schema to be used in the validation

Related reference:

- “Delimiter restrictions for moving data” on page 237
- “db2Import - Import data into a table, hierarchy, nickname or view” on page 139
- “IMPORT Command” in *Command Reference*

db2Inspect - Inspect database for architectural integrity

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

Scope:

In a single partition database environment, the scope is the single database partition only. In a partitioned database environment it is the collection of all logical database partitions defined in db2nodes.cfg. For partitioned tables, the scope for database and table space level inspection includes individual data partitions and non-partitioned indexes. Table level inspection for a partitioned table checks all the data partitions and indexes in a table, rather than checking a single data partition or index.

Authorization:

One of the following:

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

Required connection:

Database

API include file:

db2ApiDf.h

API and data structure syntax:

```
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;
```

db2Inspect - Inspect database for architectural integrity

```
    db2UInt16 iAllNodeFlag;
    db2UInt16 iNumNodes;
    db2UInt16 iLevelObjectData;
    db2UInt16 iLevelObjectIndex;
    db2UInt16 iLevelObjectLong;
    db2UInt16 iLevelObjectLOB;
    db2UInt16 iLevelObjectBlkMap;
    db2UInt16 iLevelExtentMap;
    db2UInt16 iLevelObjectXML;
} db2InspectStruct;

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;
    db2UInt16 iLevelObjectXML;
} db2gInspectStruct;
```

db2Inspect 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.

db2InspectStruct data structure parameters:

db2Inspect - Inspect database for architectural integrity

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 database partition numbers on which to perform the operation.

iAction

Input. Specifies the inspect action. Valid values (defined in the db2ApiDf header file, which is located in the include directory) 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_FORMAT_XML

Format an XML object page.

DB2INSPECT_ACT_ROWCMPEST_TBL

Estimate row compression effectiveness on a table.

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.

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

db2Inspect - Inspect database for architectural integrity

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.

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 - Inspect database for architectural integrity

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 (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_LEVEL_NORMAL

Level is normal.

DB2INSPECT_LEVEL_LOW

Level is low.

DB2INSPECT_LEVEL_NONE

Level is none.

db2Inspect - Inspect database for architectural integrity

iLevelObjectXML

Input. Specifies processing level for XML object. Valid values (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_LEVEL_NORMAL

Level is normal.

DB2INSPECT_LEVEL_LOW

Level is low.

DB2INSPECT_LEVEL_NONE

Level is none.

db2gInspectStruct data structure specific parameters:

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.

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 committing or rolling back changes, by executing a COMMIT or ROLLBACK statement respectively, 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 526
- "INSPECT command" in *Command Reference*

db2InstanceQuiesce - Quiesce instance

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 authority will have access to the database and its objects.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
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;
    db2UInt32 iTimeout;
} db2InsQuiesceStruct;

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;
```

db2InstanceQuiesce - Quiesce instance

```
        db2Uint32 iImmediate;  
        db2Uint32 iForce;  
        db2Uint32 iTimeout;  
} db2gInsQuiesceStruct;
```

db2InstanceQuiesce 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.

db2InsQuiesceStruct data structure parameters:

piInstanceName

Input. The instance name.

piUserId

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

piGroupId

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

iImmediate

Input. Valid values are:

TRUE=1

Force the applications immediately.

FALSE=0

Deferred force. Applications will wait the number of minutes specified by iTimeout parameter to let their current units of work be completed, and then will terminate. If this deferred force cannot be completed within the number of minutes specified by iTimeout parameter, the quiesce operation will fail.

iForce Input. Reserved for future use.

iTimeout

Input. Specifies the time, in minutes, to wait for applications to commit the current unit of work. If iTimeout is not specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the start_stop_timeout database manager configuration parameter will be used.

db2gInsQuiesceStruct data structure specific parameters:

iInstanceNameLen

Input. Specifies the length in bytes of piInstanceName.

iUserIdLen

Input. Specifies the length in bytes of piUserID.

iGroupIdLen

Input. Specifies the length in bytes of piGroupId.

Related reference:

db2InstanceQuiesce - Quiesce instance

- “QUIESCE command” in *Command Reference*
- “db2InstanceUnquiesce - Unquiesce instance” on page 183
- “SQLCA ” on page 526
- “db2InstanceStart - Start instance” on page 173
- “db2InstanceStop - Stop instance” on page 179

db2InstanceStart - Start instance

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

API and data structure syntax:

```
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;
```

db2InstanceStart - Start instance

```
        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;

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;

```

db2InstanceStart 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.

db2InstanceStartStruct data structure parameters:**iIsRemote**

Input. An indicator set to constant integer value TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piRemoteInstName

Input. The name of the remote instance.

piCommData

Input. A pointer to the db2DasCommData structure.

piStartOpts

Input. A pointer to the db2StartOptionsStruct structure.

db2DasCommData data structure parameters:**iCommParam**

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

piNodeOrHostName

Input. The database partition or hostname.

piUserId

Input. The user name.

piUserPw

Input. The user password.

db2StartOptionsStruct data structure parameters:**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.

db2InstanceStart - Start instance

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 header file, located in the include directory) are: are:

SQLQ_NONE

Issue the normal db2start operation.

SQLQ_ADDNODE

Issue the ADD NODE command.

SQLQ_RESTART

Issue the RESTART DATABASE command.

SQLQ_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:

SQLQ_TABLESPACES_NONE

Do not create any system temporary table spaces.

SQLQ_TABLESPACES_LIKE_NODE

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

SQLQ_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 SQLQ_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.

db2QuiesceStartStruct data structure parameters:**iIsQRequested**

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

piQUsrName

Input. The quiesced username.

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.

db2gInstanceStStruct data structure specific parameters:**iRemoteInstLen**

Input. Specifies the length in bytes of piRemoteInstName.

db2gDasCommData data structure specific parameters:**iNodeOrHostNameLen**

Input. Specifies the length in bytes of piNodeOrHostName.

iUserIdLen

Input. Specifies the length in bytes of piUserId.

iUserPwLen

Input. Specifies the length in bytes of piUserPw.

db2gQuiesceStartStruct data structure specific parameters:**iQUsrNameLen**

Input. Specifies the length in bytes of piQusrName.

iQGrpNameLen

Input. Specifies the length in bytes of piQGrpName.

Related reference:

- "START DATABASE MANAGER command" in *Command Reference*
- "db2InstanceStop - Stop instance" on page 179
- "SQLCA " on page 526
- "db2InstanceQuiesce - Quiesce instance" on page 170

Related samples:

- "instart.c -- Stop and start the current local instance (C)"

db2InstanceStart - Start instance

- "instart.C -- Stop and start the current local instance (C++)"

db2InstanceStop - Stop instance

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

API and data structure syntax:

```
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;

SQL_API_RC SQL_API_FN
db2gInstanceStop (
```

db2InstanceStop - Stop instance

```
        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;
```

db2InstanceStop 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.

db2InstanceStopStruct data structure parameters:

iIsRemote

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

piRemoteInstName

Input. The name of the remote instance.

piCommData

Input. A pointer to the db2DasCommData structure.

piStopOpts

Input. A pointer to the db2StopOptionsStruct structure.

db2DasCommData data structure parameters:

iCommParam

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

piNodeOrHostName

Input. The database partition or hostname.

piUserId

Input. The user name.

piUserPw

Input. The user password.

db2StopOptionsStruct data structure parameters:**iIsProfile**

Input. Indicates whether a profile is specified. Possible values are TRUE and FALSE. 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. Possible values are TRUE and FALSE. If specified, the stop command only affects the specified node.

iNodeNum

Input. The database partition number.

iStopOption

Input. Option. Valid values are:

SQLI_NONE

Issue the normal db2stop operation.

SQLI_FORCE

Issue the FORCE APPLICATION (ALL) command.

SQLI_DROP

Drop the node from the db2nodes.cfg file.

iCallerac

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

SQLI_DROP

Initial call. This is the default value.

SQLI_CONTINUE

Subsequent call. Continue processing after a prompt.

SQLI_TERMINATE

Subsequent call. Terminate processing after a prompt.

db2gInstanceStopStruct data structure specific parameters:**iRemoteInstLen**

Input. Specifies the length in bytes of piRemoteInstName.

db2gDasCommData data structure specific parameters:**iNodeOrHostNameLen**

Input. Specifies the length in bytes of piNodeOrHostName.

iUserIdLen

Input. Specifies the length in bytes of piUserId.

iUserPwLen

Input. Specifies the length in bytes of piUserPw.

Related reference:

- “STOP DATABASE MANAGER command” in *Command Reference*

db2InstanceStop - Stop instance

- “db2InstanceStart - Start instance” on page 173
- “SQLCA ” on page 526
- “db2InstanceQuiesce - Quiesce instance” on page 170
- “db2InstanceUnquiesce - Unquiesce instance” on page 183

Related samples:

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

db2InstanceUnquiesce - Unquiesce instance

Unquiesce all databases in the instance.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2InstanceUnquiesce (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InsUnquiesceStruct
{
    char *piInstanceName;
} db2InsUnquiesceStruct;

SQL_API_RC SQL_API_FN
db2gInstanceUnquiesce (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInsUnquiesceStruct
{
    db2UInt32 iInstanceNameLen;
    char *piInstanceName;
} db2gInsUnquiesceStruct;
```

db2InstanceUnquiesce 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.

db2InsUnquiesceStruct data structure parameters:

piInstanceName

Input. The instance name.

db2gInsUnquiesceStruct data structure specific parameters:

db2InstanceUnquiesce - Unquiesce instance

iInstanceNameLen

Input. Specifies the length in bytes of piInstanceName.

Related reference:

- “UNQUIESCE command” in *Command Reference*
- “db2InstanceQuiesce - Quiesce instance” on page 170
- “SQLCA ” on page 526
- “db2InstanceStop - Stop instance” on page 179

db2LdapCatalogDatabase - Register the database on the LDAP server

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

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```

SQL_API_RC SQL_API_FN
db2LdapCatalogDatabase (
    db2Uint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);

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

```

db2LdapCatalogDatabase 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.

db2LdapCatalogDatabaseStruct data structure parameters:**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.

db2LdapCatalogDatabase - Register the database on the LDAP server

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 tasks:

db2LdapCatalogDatabase - Register the database on the LDAP server

- “Registration of databases in the LDAP directory” in *Administration Guide: Implementation*

Related reference:

- “SQLCA ” on page 526
- “CATALOG LDAP DATABASE command” in *Command Reference*
- “db2LdapUncatalogDatabase - Deregister database from LDAP server” on page 196

db2LdapCatalogNode - Provide an alias for node name in LDAP server

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.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapCatalogNode (
    db2Uint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapCatalogNodeStruct
{
    char *piAlias;
    char *piNodeName;
    char *piBindDN;
    char *piPassword;
} db2LdapCatalogNodeStruct;
```

db2LdapCatalogNode 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.

db2LdapCatalogNodeStruct data structure parameters:

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.

db2LdapCatalogNode - Provide an alias for node name in LDAP server

Related reference:

- "SQLCA " on page 526
- "CATALOG LDAP NODE command" in *Command Reference*
- "db2LdapUncatalogNode - Delete alias for node name from LDAP server" on page 198

db2LdapDeregister - Deregister the DB2 server and cataloged databases from the LDAP server

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

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapDeregister (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2LdapDeregisterStruct
{
    char *piNodeName;
    char *piBindDN;
    char *piPassword;
} db2LdapDeregisterStruct;
```

db2LdapDeregister 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 db2LdapDeregisterStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2LdapDeregisterStruct data structure parameters:

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 tasks:

- "Deregistering the DB2 server" in *Administration Guide: Implementation*

db2LdapDeregister - Deregister the DB2 server and cataloged databases from the LDAP server

Related reference:

- “SQLCA ” on page 526
- “DEREGISTER command” in *Command Reference*
- “db2LdapRegister - Register the DB2 server on the LDAP server” on page 192

db2LdapRegister - Register the DB2 server on the LDAP server

Registers the DB2 server in LDAP (Lightweight Directory Access Protocol).

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapRegister (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapRegisterStruct
{
    char *piNodeName;
    char *piComputer;
    char *piInstance;
    unsigned short iNodeType;
    unsigned short iOsType;
    db2LdapProtocolInfo iProtocol;
    char *piComment;
    char *piBindDN;
    char *piPassword;
} db2LdapRegisterStruct;

typedef SQL_STRUCTURE db2LdapProtocolInfo
{
    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;
```

db2LdapRegister 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.

db2LdapRegister - Register the DB2 server on the LDAP server

db2LdapRegisterStruct data structure parameters:

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 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 operating systems, this is the Windows computer name. On UNIX based systems, this is the TCP/IP host name. 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

iOsType

Input. Specifies the operating system type of the server machine. If an operating system type is not specified, the local operating system type will be used for a local server and no operating system type will be used for a remote server.

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.

db2LdapProtocolInfo data structure parameters:

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_TCPIP

For TCP/IPv4 or TCP/IPv6 support

SQL_PROTOCOL_TCPIP4

For TCP/IPv4 support

db2LdapRegister - Register the DB2 server on the LDAP server

SQL_PROTOCOL_TCPIP6

For TCP/IPv6 support

SQL_PROTOCOL SOCKS

For TCP/IP with security SOCKS

SQL_PROTOCOL SOCKS4

For TCP/IPv4 with security SOCKS

SQL_PROTOCOL_NPIPE

For Windows Named Pipe support

piHostName

Input. Specify the TCP/IP host name or the IP address. The IP address can be an IPv4 or an IPv6 address. If an IP address is specified, it must match the protocol type selected. For example, if SQL_PROTOCOL_TCPIP4 is selected, the IP address specified must be an IPv4 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:

db2LdapRegister - Register the DB2 server on the LDAP server

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 tasks:

- “Registration of DB2 servers after installation” in *Administration Guide: Implementation*

Related reference:

- “SQLCA ” on page 526
- “REGISTER command” in *Command Reference*
- “db2LdapDeregister - Deregister the DB2 server and cataloged databases from the LDAP server” on page 190

db2LdapUncatalogDatabase - Deregister database from LDAP server

Removes a database entry from LDAP (Lightweight Directory Access Protocol).

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapUncatalogDatabase (
    db2Uint32 versionNumber,
    void * pParamStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUncatalogDatabaseStruct
{
    char *piAlias;
    char *piBindDN;
    char *piPassword;
} db2LdapUncatalogDatabaseStruct;
```

db2LdapUncatalogDatabase 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.

db2LdapUncatalogDatabaseStruct data structure parameters:

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 tasks:

- "Deregistering the database from the LDAP directory" in *Administration Guide: Implementation*

db2LdapUncatalogDatabase - Deregister database from LDAP server

Related reference:

- “SQLCA ” on page 526
- “UNCATALOG LDAP DATABASE command” in *Command Reference*
- “db2LdapCatalogDatabase - Register the database on the LDAP server” on page 185

db2LdapUncatalogNode - Delete alias for node name from LDAP server

Removes a node entry from LDAP (Lightweight Directory Access Protocol).

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapUncatalogNode (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUncatalogNodeStruct
{
    char *piAlias;
    char *piBindDN;
    char *piPassword;
} db2LdapUncatalogNodeStruct;
```

db2LdapUncatalogNode 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 db2LdapUncatalogNodeStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2LdapUncatalogNodeStruct data structure parameters:

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 526
- "UNCATALOG LDAP NODE command" in *Command Reference*

db2LdapUncatalogNode - Delete alias for node name from LDAP server

- “db2LdapCatalogNode - Provide an alias for node name in LDAP server” on page 188

db2LdapUpdate - Update the attributes of the DB2 server on the LDAP server

Updates the communication protocol information for the DB2 server in LDAP (Lightweight Directory Access Protocol).

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapUpdate (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUpdateStruct
{
    char *piNodeName;
    char *piComment;
    unsigned short iNodeType;
    db2LdapProtocolInfo iProtocol;
    char *piBindDN;
    char *piPassword;
} db2LdapUpdateStruct;

typedef SQL_STRUCTURE db2LdapProtocolInfo
{
    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;
```

db2LdapUpdate 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.

db2LdapUpdate - Update the attributes of the DB2 server on the LDAP server

db2LdapUpdateStruct data structure parameters:

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.

db2LdapProtocolInfo data structure parameters:

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_TCPIP

For TCP/IPv4 or TCP/IPv6 support

SQL_PROTOCOL_TCPIP4

For TCP/IPv4 support

SQL_PROTOCOL_TCPIP6

For TCP/IPv6 support

SQL_PROTOCOL SOCKS

For TCP/IP with security SOCKS

SQL_PROTOCOL SOCKS4

For TCP/IPv4 with security SOCKS

SQL_PROTOCOL_NPIPE

For Windows Named Pipe support

piHostName

Input. Specify the TCP/IP host name or the IP address. The IP address can be an IPv4 or an IPv6 address. If an IP address is specified, it must match the protocol type selected. For example, if SQL_PROTOCOL_TCPIP4 is selected, the IP address specified must be an IPv4 address.

piServiceName

Input. Specify the TCP/IP service name or port number.

db2LdapUpdate - Update the attributes of the DB2 server on the LDAP server

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.

Related reference:

- "SQLCA " on page 526
- "UPDATE LDAP NODE command" in *Command Reference*

db2LdapUpdateAlternateServerForDB - Update the alternate server for the database on the LDAP server

Updates the alternate server for a database in Lightweight Directory Access Protocol (LDAP).

Authorization:

Read/write access to the LDAP server.

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LdapUpdateAlternateServerForDB (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LdapUpdateAltServerStruct
{
    char *piDbAlias;
    char *piNode;
    char *piGWNode;
    char *piBindDN;
    char *piPassword;
} db2LdapUpdateAltServerStruct;
```

db2LdapUpdateAlternateServerForDB 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 db2LdapUpdateAltServerStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2LdapUpdateAltServerStruct data structure parameters:

piDbAlias

Input. A string containing an alias for the database to be updated.

piNode

Input. A string containing the alternate node name. This node name must exist in LDAP.

piGWNode

Input. A string containing the alternate gateway node name. This node name must exist in LDAP. This is used by the runtime client to connect to the host via the gateway.

db2LdapUpdateAlternateServerForDB - Update the alternate server for the database on the LDAP server

piBindDN

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

piPassword

Input. Account password.

Related tasks:

- "Rerouting LDAP clients to another server" in *Administration Guide: Implementation*

Related reference:

- "UPDATE ALTERNATE SERVER FOR LDAP DATABASE command" in *Command Reference*
- "db2LdapCatalogDatabase - Register the database on the LDAP server" on page 185
- "db2LdapUncatalogDatabase - Deregister database from LDAP server" on page 196
- "db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory" on page 334
- "SQLCA " on page 526

db2Load - Load data into a table

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. Although faster than the import utility, the load utility does not support loading data at the hierarchy level or loading into a nickname.

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. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

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

API and data structure syntax:

```
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;
```

db2Load - Load data into a table

```
    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 db2LoadUserExit
{
    db2Char iSourceUserExitCmd;
    struct db2Char *piInputStream;
    struct db2Char *piInputFileName;
    struct db2Char *piOutputFileName;
    db2UInt16 *piEnableParallelism;
} db2LoadUserExit;

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;
    db2UInt16 iSetIntegrityPending;
    struct db2LoadUserExit *piSourceUserExit;
} 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;

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 db2gLoadIn *piLoadInfoIn;
    struct db2LoadOut *poLoadInfoOut;
    struct db2gPartLoadIn *piPartLoadInfoIn;
    struct db2PartLoadOut *poPartLoadInfoOut;
    db2int16 iCallerAction;
    db2Uint16 iFileTypeLen;
    db2Uint16 iLocalMsgFileLen;
    db2Uint16 iTempFilesPathLen;
} db2gLoadStruct;

```

db2Load - Load data into a table

```
typedef SQL_STRUCTURE db2gLoadIn
{
    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;
    db2UInt16 iUseTablespaceLen;
    db2UInt16 iSetIntegrityPending;
    struct db2LoadUserExit *piSourceUserExit;
} db2gLoadIn;

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;
    void *piReserved1;
    db2UInt16 iHostnameLen;
    db2UInt16 iFileTransferLen;
    db2UInt16 iPartFileLocLen;
    db2UInt16 iMapFileInputLen;
    db2UInt16 iMapFileOutputLen;
    db2UInt16 iDistfileLen;
} db2gPartLoadIn;
```

db2Load 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.

db2LoadStruct data structure parameters:

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` header file, located in the `include` directory) 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.

SQLU_REMOTEFETCH

If the `media_type` field is set to this value, the caller provides information through an `sqlu_remotefetch_entry` structure. The `sessions` field must be set to the value of 1.

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.

db2Load - Load data into a table

The information provided in this structure depends on the value of the `media_type` field. Valid values (defined in `sqlutil` header file, located in the `include` directory) 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 tbnam [(column_list)]
[DATA LINK SPECIFICATION data link-spec]
[FOR EXCEPTION e_tbnam]"
```

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. The exception table is used to store copies of rows that violate unique index rules, range constraints and security policies.

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

db2Load - Load data into a table

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 either of type `SQLU_SQL_STMT` or `SQLU_REMOTEFETCH`, and refers to an SQL query or a table name.

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. See related link "File type modifiers for the load utility."

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 parameter (defined in `sqlutil` header file, located in the include directory) 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 header file, located in the include directory) 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.

db2Load - Load data into a table

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.

db2LoadUserExit data structure parameters:

iSourceUserExitCmd

Input. The fully qualified name of an executable that will be used to feed data to the utility. For security reasons, the executable must be placed within the `sqllib/bin` directory on the server. This parameter is mandatory if the `piSourceUserExit` structure is not NULL.

The `piInputStream`, `piInputFileName`, `piOutputFileName` and `piEnableParallelism` fields are optional. See the Data Movement Utilities Guide for a detailed description.

piInputStream

Input. A generic byte-stream that will be passed directly to the user-exit application via STDIN. You have complete control over what data is contained in this byte-stream and in what format. The load utility will simply carry this byte-stream over to the server and pass it into the user-exit application by feeding the process' STDIN (it will not codepage convert or modify the byte-stream). Your user-exit application would read the arguments from STDIN and use the data however intended.

One important attribute of this feature is the ability to hide sensitive information (such as userid/passwords). See the Data Movement Utilities Guide for a detailed description.

piInputFileName

Input. Contains the name of a fully qualified client-side file, whose contents will be passed into the user-exit application by feeding the process' STDIN.

piOutputFileName

Input. The fully qualified name of a server-side file. The STDOUT and STDERR streams of the process which is executing the user-exit application will be streamed into this file. When `piEnableParallelism` is RUE, multiple files will be created (one per user-exit instance), and each file name will be appended with a 3 digit numeric node-number value, such as `<filename>.000`.

piEnableParallelism

Input. A flag indicating that the utility should attempt to parallelize the invocation of the user-exit application. See the Data Movement Utilities Guide for a detailed description.

db2LoadIn data structure parameters:

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 the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If `warningcnt` is 0, or this option is not specified, the load operation will continue regardless of the number of warnings issued.

db2Load - Load data into a table

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` header file, located in the include directory) 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

This parameter is being deprecated as of Version 9.1. Use the iSetIntegrityPending parameter instead.

iRestartphase

Input. Reserved. Valid value is a single space character ' '.

iStatsOpt

Input. Granularity of statistics to collect. Valid values are:

SQLU_STATS_NONE

No statistics to be gathered.

SQLU_STATS_USE_PROFILE

Statistics are collected based on the profile defined for the current table. This profile must be created using the RUNSTATS command. If no profile exists for the current table, a warning is returned and no statistics are collected.

iSetIntegrityPending

Input. Specifies to put the table into set integrity pending state. If the value SQLU_SI_PENDING_CASCADE_IMMEDIATE is specified, set integrity pending state will be immediately cascaded to all dependent and descendent tables. If the value SQLU_SI_PENDING_CASCADE_DEFERRED is specified, the cascade of set integrity pending state to dependent tables will be deferred until the target table is checked for integrity violations. SQLU_SI_PENDING_CASCADE_DEFERRED is the default value if the option is not specified.

piSourceUserExit

Input. A pointer to the db2LoadUserExit structure.

db2LoadOut data structure parameters:**oRowsRead**

Output. Number of records read during the load operation.

db2Load - Load data into a table

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.

db2PartLoadIn data structure parameters:

piHostname

Input. The hostname for the iFileTransferCmd parameter. If NULL, the hostname will default to "nohost". This parameter is deprecated.

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. This parameter is deprecated. Use the piSourceUserExit parameter instead.

piPartFileLocation

Input. In PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the partitioned files. This location must exist on each database partition specified by the piOutputNodes option.

For the SQL_CURSOR file type, this parameter cannot be 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 database partition for PARTITION_ONLY mode, or the location of the files to be read from each database 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. For file types other than SQL_CURSOR, if the value of this parameter is NULL, it will default to the current directory.

piOutputNodes

Input. The list of Load output database 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 header file, located in the include directory) are:

- DB2LOAD_PARTITION_AND_LOAD

Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

- DB2LOAD_PARTITION_ONLY

Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than SQL_CURSOR, the name of the output file on each database partition will have the form filename.xxx, where filename is the name of the first input file specified by piSourceList and xxx is the database partition number. For the SQL_CURSOR file type, the name of the output file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY

Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than SQL_CURSOR, the input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number. For the SQL_CURSOR file type, the name of the input file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY_VERIFY_PART

Data is assumed to be already distributed, but the data file does not contain a database partition header. The distribution 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 database partition. Rows containing database partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning will be written to the load message file for that database partition. The input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_ANALYZE

An optimal distribution 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.

db2Load - Load data into a table

piIsolatePartErrs

Input. Indicates how the load operation will react to errors that occur on individual database partitions. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOAD_SETUP_ERRS_ONLY

In this mode, errors that occur on a database partition during setup, such as problems accessing a database partition or problems accessing a table space or table on a database partition, will cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each database partition.

DB2LOAD_LOAD_ERRS_ONLY

In this mode, errors that occur on a database partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the database partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining database partitions until a failure occurs or until all the data is loaded. On the database 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 database partitions the transaction will be aborted. Data on all of the database partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the database partitions where the load operation completed and resume the load operation on database partitions that experienced an error.

Note: This mode cannot be used when `iAccessLevel` is set to `SQLU_ALLOW_READ_ACCESS` and a copy target is also specified.

DB2LOAD_SETUP_AND_LOAD_ERRS

In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the `DB2LOAD_LOAD_ERRS_ONLY` mode, when database partition errors do occur while data is being loaded, the data on all database 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` and a copy target is also 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` and a copy target is also 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. Distribution 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. Distribution 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 RECLLEN file type modifier is also specified. Possible values are TRUE and FALSE. If NULL, the value defaults to FALSE.

piDistfile

Input. Name of the database partition distribution file. If a NULL is specified, the value defaults to "DISTFILE".

piOmitHeader

Input. Indicates that a distribution map header should not be included in the database 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 database partition list.

db2LoadNodeList data structure parameters:**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.

db2LoadPortRange data structure parameters:**iPortMin**

Input. Lower port number.

iPortMax

Input. Higher port number.

db2Load - Load data into a table

db2PartLoadOut data structure parameters:

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-partitioning 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 database 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 database 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 database 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.

db2LoadAgentInfo data structure parameters:

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 database 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 SET INTEGRITY PENDING state due to the need for further constraints processing, but this will not be 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 database 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 database partitions.

DB2LOADQUERY_LOADPENDING

Indicates that the load job aborted during processing but left the table on the database partition in the LOAD PENDING state, indicating that the load job on that database partition must be either terminated or restarted.

oNodeNum

Output. The number of the database partition on which the agent executed.

oAgentType

Output. The agent type. Valid values (defined in db2ApiDf header file, located in the include directory) are :

- DB2LOAD_LOAD_AGENT
- DB2LOAD_PARTITIONING_AGENT
- DB2LOAD_PRE_PARTITIONING_AGENT
- DB2LOAD_FILE_TRANSFER_AGENT
- DB2LOAD_LOAD_TO_FILE_AGENT

db2gLoadStruct data structure specific parameters:**iFileTypeLen**

Input. Specifies the length in bytes of iFileType parameter.

iLocalMsgFileLen

Input. Specifies the length in bytes of iLocalMsgFileName parameter.

iTempFilesPathLen

Input. Specifies the length in bytes of iTempFilesPath parameter.

db2gLoadIn data structure specific parameters:**iUseTablespaceLen**

Input. The length in bytes of piUseTablespace parameter.

db2gPartLoadIn data structure specific parameters:**piReserved1**

Reserved for future use.

iHostnameLen

Input. The length in bytes of piHostname parameter.

db2Load - Load data into a table

iFileTransferLen

Input. The length in bytes of piFileTransferCmd parameter.

iPartFileLocLen

Input. The length in bytes of piPartFileLocation parameter.

iMapFileInputLen

Input. The length in bytes of piMapFileInput parameter.

iMapFileOutputLen

Input. The length in bytes of piMapFileOutput parameter.

iDistfileLen

Input. The length in bytes of piDistfile parameter.

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 set integrity pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in set integrity pending state. Issue the SET INTEGRITY statement to take the tables out of set integrity 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.

Related tasks:

- “Loading data” in *Data Movement Utilities Guide and Reference*

Related reference:

- “LOAD command” in *Command Reference*
- “sqldcol ” on page 530
- “sqlu_media_list ” on page 582
- “db2LoadQuery - Get the status of a load operation” on page 239
- “db2Export - Export data from a database” on page 81
- “db2Import - Import data into a table, hierarchy, nickname or view” on page 139

Related samples:

- “dtformat.sqc -- Load and import data format extensions (C)”
- “tblog.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++)”

File type modifiers for the load utility

Table 16. Valid file type modifiers for the load utility: All file formats

Modifier	Description
anyorder	This modifier is used in conjunction with the <code>cpu_parallelism</code> parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of <code>cpu_parallelism</code> is 1, this option is ignored. This option is not supported if <code>SAVECOUNT > 0</code> , since crash recovery after a consistency point requires that data be loaded in sequence.
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 <code>generatedmissing</code> or the <code>generatedoverride</code> 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 <code>generatedignore</code> or the <code>generatedoverride</code> modifier.
generatedoverride	<p>This modifier instructs the load utility to accept 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 <code>RECOVER DROPPED TABLE</code> option on the <code>ROLLFORWARD DATABASE</code> command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W). When this modifier is used, the table will be placed in Set Integrity Pending state. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command after the load operation:</p> <pre>SET INTEGRITY FOR < table-name > GENERATED COLUMN IMMEDIATE UNCHECKED</pre> <p>To take the table out of Set Integrity Pending state and force verification of the user-supplied values, issue the following command after the load operation:</p> <pre>SET INTEGRITY FOR < table-name > IMMEDIATE CHECKED.</pre> <p>When this modifier is specified and there is a generated column in any of the partitioning keys, dimension keys or distribution keys, then the LOAD command will automatically convert the modifier to <code>generatedignore</code> and proceed with the load. This will have the effect of regenerating all of the generated column values.</p> <p>This modifier cannot be used with either the <code>generatedmissing</code> or the <code>generatedignore</code> modifier.</p>
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 <code>GENERATED ALWAYS</code> and <code>GENERATED BY DEFAULT</code> identity columns. This means that for <code>GENERATED ALWAYS</code> columns, no rows will be rejected. This modifier cannot be used with either the <code>identitymissing</code> or the <code>identityoverride</code> 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 <code>GENERATED ALWAYS</code> and <code>GENERATED BY DEFAULT</code> identity columns. This modifier cannot be used with either the <code>identityignore</code> or the <code>identityoverride</code> modifier.

File type modifiers for the load utility

Table 16. Valid file type modifiers for the load utility: All file formats (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. 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= <i>x</i>	<p><i>x</i> 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 load rebuilds the index. Load with INDEXING MODE INCREMENTAL ignores this option. The first entry in a page is added without restriction; subsequent entries are added the percent free space threshold can be maintained. The default value is the one used at CREATE INDEX time.</p> <p>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.</p>
lobsinfile	<p><i>lob-path</i> specifies the path to the files containing LOB data. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column.</p> <p>This option is not supported in conjunction with the CURSOR filetype.</p> <p>The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the LOAD utility the list of paths to search for the LOB files while loading the data.</p> <p>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 <i>filename.ext.nnn.mmm/</i>, where <i>filename.ext</i> is the name of the file that contains the LOB, <i>nnn</i> is the offset in bytes of the LOB within the file, and <i>mmm</i> is the length of the LOB in bytes. For example, if the string <i>db2exp.001.123.456/</i> is stored in the data file, the LOB is located at offset 123 in the file <i>db2exp.001</i>, and is 456 bytes long.</p> <p>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 <i>db2exp.001.7.-1/</i>.</p>
noheader	<p>Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group).</p> <p>If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single-partition database partition group, the file is not expected to have a header. Thus the noheader modifier is not needed. If the LOAD_ONLY mode is used, the file is expected to have a header. The only circumstance in which you should need to use the noheader modifier is if you wanted to perform LOAD_ONLY operation using a file that does not have a header.</p>
norowwarnings	Suppresses all warnings about rejected rows.

Table 16. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
pagefreespace= <i>x</i>	<p><i>x</i> 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 3 000 bytes long, and an <i>x</i> 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. 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.</p>
seclabelchar	<p>Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. LOAD converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3242W) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.</p> <p>This modifier cannot be specified if the seclabelname modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.</p> <p>If you have a table consisting of a single DB2SECURITYLABEL column, the data file might look like this:</p> <pre> "CONFIDENTIAL:ALPHA:G2" "CONFIDENTIAL;SIGMA:G2" "TOP SECRET:ALPHA:G2" </pre> <p>To load or import this data, the SECLABELCHAR file type modifier must be used:</p> <pre>LOAD FROM input.del OF DEL MODIFIED BY SECLABELCHAR INSERT INTO t1</pre>
seclabelname	<p>Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. LOAD will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.</p> <p>This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.</p> <p>If you have a table consisting of a single DB2SECURITYLABEL column, the data file might consist of security label names similar to:</p> <pre> "LABEL1" "LABEL1" "LABEL2" </pre> <p>To load or import this data, the SECLABELNAME file type modifier must be used:</p> <pre>LOAD FROM input.del OF DEL MODIFIED BY SECLABELNAME INSERT INTO t1</pre> <p>Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.</p>

File type modifiers for the load utility

Table 16. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
totalreespace= <i>x</i>	<p><i>x</i> 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 <i>x</i> 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. If two loads are done with this option specified, the second load will not reuse the extra space appended to the end by the first load.</p>
usedefaults	<p>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:</p> <ul style="list-style-type: none"> • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (" , ") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. <p>Without this option, if a source column contains no data for a row instance, one of the following occurs:</p> <ul style="list-style-type: none"> • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 17. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage= <i>x</i>	<p><i>x</i> 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.</p> <p>The following rules apply:</p> <ul style="list-style-type: none"> • 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 might 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. <p>This option is not supported in conjunction with the CURSOR filetype.</p>

Table 17. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
dateformat="x"	<p>x is the format of the date in the source file.¹ Valid date elements are:</p> <p>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)</p> <p>A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:</p> <p>"D-M-YYYY" "MM.DD.YYYY" "YYYYDDD"</p>
dumpfile = x	<p>x is the fully qualified (according to the server database partition) name of an exception file to which rejected rows are written. A maximum of 32 KB of data is written per record. Following is an example that shows how to specify a dump file:</p> <pre>db2 load from data of del modified by dumpfile = /u/user/filename insert into table_name</pre> <p>The file will be created and owned by the instance owner. To override the default file permissions, use the dumpfileaccessall file type modifier.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. In a partitioned database environment, the path should be local to the loading database partition, 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.
dumpfileaccessall	<p>Grants read access to 'OTHERS' when a dump file is created.</p> <p>This file type modifier is only valid when:</p> <ol style="list-style-type: none"> 1. it is used in conjunction with dumpfile file type modifier 2. the user has SELECT privilege on the load target table 3. it is issued on a DB2 server database partition that resides on a UNIX operating system
fastparse	<p>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.</p>
implieddecimal	<p>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, <i>not</i> 12345.00.</p> <p>This modifier cannot be used with the packeddecimal modifier.</p>

File type modifiers for the load utility

Table 17. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
timeformat="x"	<p>x is the format of the time in the source file.¹ Valid time elements are:</p> <ul style="list-style-type: none"> 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) <p>A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:</p> <pre>"HH:MM:SS" "HH.MM TT" "SSSSS"</pre>

Table 17. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
timestampformat="x"	<p>x is the format of the time stamp in the source file.¹ Valid time stamp elements are:</p> <p>YYYY - Year (four digits ranging from 0000 - 9999)</p> <p>M - Month (one or two digits ranging from 1 - 12)</p> <p>MM - Month (two digits ranging from 01 - 12; mutually exclusive with M and MMM)</p> <p>MMM - Month (three-letter case-insensitive abbreviation for the month name; mutually exclusive with M and MM)</p> <p>D - Day (one or two digits ranging from 1 - 31)</p> <p>DD - Day (two digits ranging from 1 - 31; mutually exclusive with D)</p> <p>DDD - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)</p> <p>H - Hour (one or two digits ranging from 0 - 12 for a 12 hour system, and 0 - 24 for a 24 hour system)</p> <p>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)</p> <p>M - Minute (one or two digits ranging from 0 - 59)</p> <p>MM - Minute (two digits ranging from 0 - 59; mutually exclusive with M, minute)</p> <p>S - Second (one or two digits ranging from 0 - 59)</p> <p>SS - Second (two digits ranging from 0 - 59; mutually exclusive with S)</p> <p>SSSSS - Second of the day after midnight (5 digits ranging from 00000 - 86399; mutually exclusive with other time elements)</p> <p>UUUUUU - Microsecond (6 digits ranging from 000000 - 999999; mutually exclusive with all other microsecond elements)</p> <p>UUUUU - Microsecond (5 digits ranging from 00000 - 99999, maps to range from 000000 - 999990; mutually exclusive with all other microsecond elements)</p> <p>UUUU - Microsecond (4 digits ranging from 0000 - 9999, maps to range from 000000 - 999900; mutually exclusive with all other microsecond elements)</p> <p>UUU - Microsecond (3 digits ranging from 000 - 999, maps to range from 000000 - 999000; mutually exclusive with all other microsecond elements)</p> <p>UU - Microsecond (2 digits ranging from 00 - 99, maps to range from 000000 - 990000; mutually exclusive with all other microsecond elements)</p> <p>U - Microsecond (1 digit ranging from 0 - 9, maps to range from 000000 - 900000; mutually exclusive with all other microsecond elements)</p> <p>TT - Meridian indicator (AM or PM)</p> <p>A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of 'Jan' is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:</p> <pre>"YYYY/MM/DD HH:MM:SS.UUUUUU"</pre> <p>The valid values for the MMM element include: 'jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov' and 'dec'. These values are case insensitive.</p> <p>The following example illustrates how to import data containing user defined date and time formats into a table called schedule:</p> <pre>db2 import from delfile2 of del modified by timestampformat="yyyy.mm.dd hh:mm tt" insert into schedule</pre>

File type modifiers for the load utility

Table 17. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	<p>If usegraphiccodepage is given, the assumption is made that data being loaded into graphic or double-byte character large object (DBCLOB) data field(s) is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic codepage is associated with the character code page. LOAD determines the character code page through either the codepage modifier, if it is specified, or through the code page of the database if the codepage modifier is not specified.</p> <p>This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data.</p> <p>Restrictions</p> <p>The usegraphi ccodepage modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The usegraphi ccodepage modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.</p>

Table 18. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII)

Modifier	Description
binarynumerics	<p>Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions.</p> <p>This option is supported only with positional ASC, using fixed length records specified by the reclen option.</p> <p>The following rules apply:</p> <ul style="list-style-type: none"> • 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. <p>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.</p>
nochecklengths	<p>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.</p>
nullindchar=x	<p>x is a single character. Changes the character denoting a NULL value to x. The default value of x is Y.²</p> <p>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.</p>

Table 18. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description
packeddecimal	<p>Loads packed-decimal data directly, since the <code>binarynumerics</code> modifier does not include the <code>DECIMAL</code> field type.</p> <p>This option is supported only with positional ASC, using fixed length records specified by the <code>reclen</code> option.</p> <p>Supported values for the sign nibble are:</p> <ul style="list-style-type: none"> + = 0xC 0xA 0xE 0xF - = 0xD 0xB <p>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.</p> <p>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 Windows operating systems, the byte order must not be reversed.</p> <p>This modifier cannot be used with the <code>implieddecimal</code> modifier.</p>
reclen= <i>x</i>	<p><i>x</i> is an integer with a maximum value of 32 767. <i>x</i> characters are read for each row, and a new-line character is not used to indicate the end of the row.</p>
striptblanks	<p>Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.</p> <p>This option cannot be specified together with <code>striptnulls</code>. These are mutually exclusive options. This option replaces the obsolete <code>t</code> option, which is supported for back-level compatibility only.</p>
striptnulls	<p>Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept.</p> <p>This option cannot be specified together with <code>striptblanks</code>. These are mutually exclusive options. This option replaces the obsolete <code>padwithzero</code> option, which is supported for back-level compatibility only.</p>
zoneddecimal	<p>Loads zoned decimal data, since the <code>BINARYNUMERICS</code> modifier does not include the <code>DECIMAL</code> field type. This option is supported only with positional ASC, using fixed length records specified by the <code>RECLLEN</code> option.</p> <p>Half-byte sign values can be one of the following:</p> <ul style="list-style-type: none"> + = 0xC 0xA 0xE 0xF - = 0xD 0xB <p>Supported values for digits are 0x0 to 0x9.</p> <p>Supported values for zones are 0x3 and 0xF.</p>

File type modifiers for the load utility

Table 19. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII)

Modifier	Description
chardelx	<p>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.²³ If you wish to explicitly specify the double quotation mark(") as the character string delimiter, you should specify it as follows:</p> <pre>modified by chardel""</pre> <p>The single quotation mark (') can also be specified as a character string delimiter as follows:</p> <pre>modified by chardel''</pre>
coldelx	<p>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.²³</p>
decplusblank	<p>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.</p>
decptx	<p>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.²³</p>
delprioritychar	<p>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:</p> <pre>db2 load ... modified by delprioritychar ...</pre> <p>For example, given the following DEL data file:</p> <pre>"Smith, Joshua",4000,34.98<row delimiter> "Vincent,<row delimiter>, is a manager", 4005,44.37<row delimiter></pre> <p>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 <i>not</i> specified, there will be three rows in this data file, each delimited by a <row delimiter>.</p>
keepblanks	<p>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.</p> <p>The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file:</p> <pre>db2 load from delfile3 of del modified by keepblanks insert into table1</pre>
nochardel	<p>The load utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption.</p> <p>This option cannot be specified with chardelx, delprioritychar or nodoubledel. These are mutually exclusive options.</p>
nodoubledel	<p>Suppresses recognition of double character delimiters.</p>

Table 20. Valid file type modifiers for the load utility: IXF file format

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.

Notes:

1. 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)
```

Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).

2. 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 ...
```
3. Delimiter restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.

File type modifiers for the load utility

4. 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.

Table 21. LOAD behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Absent	All data in the file is assumed to be in the database code page, not the application code page, even if the CLIENT option is specified.
Present	Absent	All data in the file is assumed to be in code page N. Warning: Graphic data will be corrupted when loaded into the database if N is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the database code page, even if the CLIENT option is specified. Graphic data is assumed to be in the code page of the database graphic data, even if the CLIENT option is specified. If the database code page is single-byte, then all data is assumed to be in the database code page. Warning: Graphic data will be corrupted when loaded into a single-byte database.
Present	Present	Character data is assumed to be in code page N. Graphic data is assumed to be in the graphic code page of N. If N is a single-byte or double-byte code page, then all data is assumed to be in code page N. Warning: Graphic data will be corrupted when loaded into the database if N is a single-byte code page.

Related reference:

- “db2Load - Load data into a table” on page 205
- “Delimiter restrictions for moving data” on page 237
- “LOAD command” in *Command Reference*

Delimiter restrictions for moving data

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 might 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 charde10x0C colde1X1e ...

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 reference:

- "File type modifiers for the export utility" on page 89

Delimiter restrictions for moving data

- “File type modifiers for the import utility” on page 153
- “File type modifiers for the load utility” on page 225

db2LoadQuery - Get the status of a load operation

Checks the status of a load operation during processing.

Authorization:

None

Required connection:

Database

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2LoadQuery (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadQueryStruct
{
    db2Uint32 iStringType;
    char *piString;
    db2Uint32 iShowLoadMessages;
    struct db2LoadQueryOutputStruct *poOutputStruct;
    char *piLocalMessageFile;
} db2LoadQueryStruct;

typedef SQL_STRUCTURE db2LoadQueryOutputStruct
{
    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;

typedef SQL_STRUCTURE db2LoadQueryOutputStruct64
{
    db2Uint64 oRowsRead;
    db2Uint64 oRowsSkipped;
    db2Uint64 oRowsCommitted;
    db2Uint64 oRowsLoaded;
    db2Uint64 oRowsRejected;
    db2Uint64 oRowsDeleted;
    db2Uint32 oCurrentIndex;
    db2Uint32 oNumTotalIndexes;
    db2Uint32 oCurrentMPPNode;
    db2Uint32 oLoadRestarted;
    db2Uint32 oWhichPhase;
    db2Uint32 oWarningCount;
    db2Uint32 oTableState;
```

db2LoadQuery - Get the status of a load operation

```
} db2LoadQueryOutputStruct64;

typedef SQL_STRUCTURE db2LoadQueryStruct64
{
    db2UInt32 iStringType;
    char *piString;
    db2UInt32 iShowLoadMessages;
    struct db2LoadQueryOutputStruct64 *poOutputStruct;
    char *piLocalMessageFile;
} db2LoadQueryStruct64;

SQL_API_RC SQL_API_FN
db2gLoadQuery (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadQueryStruct
{
    db2UInt32 iStringType;
    db2UInt32 iStringLen;
    char *piString;
    db2UInt32 iShowLoadMessages;
    struct db2LoadQueryOutputStruct *poOutputStruct;
    db2UInt32 iLocalMessageFileLen;
    char *piLocalMessageFile;
} db2gLoadQueryStruct;

typedef SQL_STRUCTURE db2gLoadQueryStru64
{
    db2UInt32 iStringType;
    db2UInt32 iStringLen;
    char *piString;
    db2UInt32 iShowLoadMessages;
    struct db2LoadQueryOutputStruct64 *poOutputStruct;
    db2UInt32 iLocalMessageFileLen;
    char *piLocalMessageFile;
} db2gLoadQueryStru64;
```

db2LoadQuery 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. If the version is Version 9 or higher, it is a pointer to the db2LoadQueryStruct64 structure. Otherwise, it is a pointer to the db2LoadQueryStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2LoadQueryStruct data structure parameters:

iStringType

Input. Specifies a type for piString. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_TABLENAME

Specifies a table name for use by the db2LoadQuery API.

piString

Input. Specifies a temporary files path name or a table name, depending on the value of iStringType.

db2LoadQuery - Get the status of a load operation

iShowLoadMessages

Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

piLocalMessageFile

Input. Specifies the name of a local file to be used for output messages.

db2LoadQueryOutputStruct data structure parameters:

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).

oNumTotalIndexes

Output. Total number of indexes to be 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 header file, located in the include directory) are:

db2LoadQuery - Get the status of a load operation

DB2LOADQUERY_LOAD_PHASE

Load phase.

DB2LOADQUERY_BUILD_PHASE

Build phase.

DB2LOADQUERY_DELETE_PHASE

Delete phase.

DB2LOADQUERY_INDEXCOPY_PHASE

Index copy phase.

oWarningCount

Output. Total number of warnings returned so far.

oTableState

Output. The table states. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_NORMAL

No table states affect the table.

DB2LOADQUERY_SI_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_SI_PENDING state. The load utility puts a table into the DB2LOADQUERY_SI_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_REORG_PENDING

A reorg recommended alter has been performed on this table. A classic reorg must be performed before the table will be accessible.

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.

db2LoadQuery - Get the status of a load operation

DB2LOADQUERY_TYPE1_INDEXES

The table currently uses type-1 indexes. The indexes can be converted to type-2 using the CONVERT option when using the REORG utility on the indexes.

db2LoadQueryOutputStruct64 data structure parameters:

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).

oNumTotalIndexes

Output. Total number of indexes to be 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 header file, located in the include directory) are:

DB2LOADQUERY_LOAD_PHASE

Load phase.

DB2LOADQUERY_BUILD_PHASE

Build phase.

DB2LOADQUERY_DELETE_PHASE

Delete phase.

DB2LOADQUERY_INDEXCOPY_PHASE

Index copy phase.

oWarningCount

Output. Total number of warnings returned so far.

oTableState

Output. The table states. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2LoadQuery - Get the status of a load operation

DB2LOADQUERY_NORMAL

No table states affect the table.

DB2LOADQUERY_SI_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_SI_PENDING state. The load utility puts a table into the DB2LOADQUERY_SI_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_REORG_PENDING

A reorg recommended alter has been performed on this table. A classic reorg must be performed before the table will be accessible.

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.

DB2LOADQUERY_TYPE1_INDEXES

The table currently uses type-1 indexes. The indexes can be converted to type-2 using the CONVERT option when using the REORG utility on the indexes.

db2LoadQueryStruct64 data structure parameters:

iStringType

Input. Specifies a type for piString. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_TABLENAME

Specifies a table name for use by the db2LoadQuery API.

piString

Input. Specifies a temporary files path name or a table name, depending on the value of iStringType.

db2LoadQuery - Get the status of a load operation

iShowLoadMessages

Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

piLocalMessageFile

Input. Specifies the name of a local file to be used for output messages.

db2gLoadQueryStruct data structure specific parameters:

iStringLen

Input. Specifies the length in bytes of piString parameter.

iLocalMessageFileLen

Input. Specifies the length in bytes of piLocalMessageFile parameter.

db2gLoadQueryStru64 data structure specific parameters:

iStringLen

Input. Specifies the length in bytes of piString parameter.

iLocalMessageFileLen

Input. Specifies the length in bytes of piLocalMessageFile parameter.

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 piLocalMessageFile.

Related concepts:

- “Monitoring a load operation in a partitioned database environment using the LOAD QUERY command” in *Data Movement Utilities Guide and Reference*

Related reference:

- “LOAD QUERY command” in *Command Reference*
- “SQLCA ” on page 526
- “db2Load - Load data into a table” on page 205

Related samples:

- “loadqry.sqb -- Query the current status of a load (MF COBOL)”
- “tblog.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 or update the monitor switch settings

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
- sysmon

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2MonitorSwitches (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2MonitorSwitchesData
{
    struct sqlm_recording_group *piGroupStates;
    void *poBuffer;
    db2UInt32 iBufferSize;
    db2UInt32 iReturnData;
    db2UInt32 iVersion;
    db2int32 iNodeNumber;
    db2UInt32 *poOutputFormat;
} db2MonitorSwitchesData;
```

```
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;
```

db2MonitorSwitches - Get or update the monitor switch settings

```
db2UInt32 iBufferSize;  
db2UInt32 iReturnData;  
db2UInt32 iVersion;  
db2int32 iNodeNumber;  
db2UInt32 *poOutputFormat;  
} db2gMonitorSwitchesData;
```

db2MonitorSwitches API parameters:

versionNumber

Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct. To use the structure as described above, specify db2Version810. If you want to use a different version of this structure, check the db2ApiDf.h header file in the include directory for the complete list of supported versions. Ensure that you use the version of the db2MonitorSwitchesStruct structure that corresponds to the version number that you specify.

pParmStruct

Input. A pointer to the db2MonitorSwitchesStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2MonitorSwitchesData data structure parameters:

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:

db2MonitorSwitches - Get or update the monitor switch settings

- 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 tasks:

- “Setting monitor switches from a client application” in *System Monitor Guide and Reference*

Related reference:

- “GET DATABASE MANAGER MONITOR SWITCHES command” in *Command Reference*
- “GET MONITOR SWITCHES command” in *Command Reference*
- “UPDATE MONITOR SWITCHES command” in *Command Reference*
- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API” on page 117
- “db2ResetMonitor - Reset the database system monitor data” on page 281
- “SQLCA ” on page 526
- “DBMCFG administrative view – Retrieve database manager configuration parameter information” in *Administrative SQL Routines and Views*

Related samples:

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

db2Prune - Delete the history file entries or log files from the active log path

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Prune (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2PruneStruct
{
    char *piString;
    db2HistoryEID iEID;
    db2UInt32 iAction;
    db2UInt32 iOptions;
} db2PruneStruct;

SQL_API_RC SQL_API_FN
db2gPrune (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gPruneStruct
{
    db2UInt32 iStringLen;
    char *piString;
    db2HistoryEID iEID;
    db2UInt32 iAction;
    db2UInt32 iOptions;
} db2gPruneStruct;
```

db2Prune API parameters:

versionNumber

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

db2Prune - Delete the history file entries or log files from the active log path

pParmStruct

Input. A pointer to the db2PruneStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2PruneStruct data structure parameters:

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; a NULL parameter value is invalid.

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.

iAction

Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) 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 header file, located in the include directory) are:

DB2PRUNE_OPTION_FORCE

Force the removal of the last backup.

DB2PRUNE_OPTION_DELETE

Delete log files that are pruned from the history file.

DB2PRUNE_OPTION_LSNSTRING

Specify that the value of piString is an LSN, used when a caller action of DB2PRUNE_ACTION_LOG is specified.

db2gPruneStruct data structure specific parameters:

iStringLen

Input. Specifies the length in bytes of piString.

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.

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.

db2Prune - Delete the history file entries or log files from the active log path

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.

Related reference:

- “PRUNE HISTORY/LOGFILE command” in *Command Reference*
- “db2HistoryCloseScan - End the history file scan” on page 127
- “db2HistoryGetEntry - Get the next entry in the history file” on page 129
- “db2HistoryOpenScan - Start a history file scan” on page 132
- “db2HistoryUpdate - Update a history file entry” on page 136
- “SQLCA ” on page 526
- “ADMIN_CMD procedure – Run administrative commands” in *Administrative SQL Routines and Views*

Related samples:

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

db2QuerySatelliteProgress - Get the status of a satellite synchronization session

Checks on the status of a satellite synchronization session.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2QuerySatelliteProgress (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef struct db2QuerySatelliteProgressStruct
{
    db2int32 oStep;
    db2int32 oSubstep;
    db2int32 oNumSubsteps;
    db2int32 oScriptStep;
    db2int32 oNumScriptSteps;
    char *poDescription;
    char *poError;
    char *poProgressLog;
} db2QuerySatelliteProgressStruct;
```

db2QuerySatelliteProgress 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.

db2QuerySatelliteProgressStruct data structure parameters:

oStep Output. The current step of the synchronization session (defined in db2ApiDf header file, located in the include directory).

oSubstep

Output. If the synchronization step indicated by parameter, 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.

db2QuerySatelliteProgress - Get the status of a satellite synchronization session

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 526
- "db2GetSyncSession - Get a satellite synchronization session identifier" on page 120
- "db2SyncSatellite - Start satellite synchronization" on page 325
- "db2SetSyncSession - Set satellite synchronization session" on page 318
- "db2SyncSatelliteStop - Pause satellite synchronization" on page 326
- "db2SyncSatelliteTest - Test whether a satellite can be synchronized" on page 327

db2ReadLog - Extracts log records

Extracts log records from the DB2 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 the database configuration parameters logarchmeth1 and/or logarchmeth2 are not set to OFF.

Authorization:

One of the following:

- sysadm
- dbadm

Required connection:

Database

API include file:

db2ApiDf.h

API and data structure syntax:

```
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;
} db2ReadLogStruct;

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;
```

db2ReadLog 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 structure.

pSqlca

Output. A pointer to the sqlca structure.

db2ReadLogStruct data structure parameters:**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 the startLsn parameter, 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 behavior of the asynchronous log read API. The log records that are returned when this value is

db2ReadLog - Extracts log records

used are documented in the "DB2 log records" topic. All other log records are for IBM internal use only and are therefore not documented.

poReadLogInfo

Output. A structure detailing information regarding the call and the database log.

db2ReadLogInfoStruct data structure parameters:

initialLSN

The first LSN used, or that will be used, by the database since it was activated.

firstReadLSN

The first LSN present in poLogBuffer parameter.

nextStartLSN

The start of the next log record the caller should read. Because some log records can be filtered and not returned in poLogBuffer parameter, using this LSN as the start of the next read instead of the end of the last log record in poLogBuffer parameter will prevent rescanning log records which have already been filtered.

logRecsWritten

The number of log records written to poLogBuffer parameter.

logBytesWritten

The total number of bytes of data written to poLogBuffer parameter.

firstReusedLSN

The first LSN to be reused due to a database restore or rollforward operation.

timeOfLSNReuse

The time at which the LSN represented by firstReusedLSN was reused. The time is the number of seconds since January 1, 1970.

currentTimeValue

The current time according to the database.

db2TimeOfLog data structure parameters:

seconds

The number of seconds since January 1, 1970.

accuracy

A high accuracy counter which allows callers to distinguish the order of events when comparing timestamps that occurred within the same second.

Usage notes:

If the requested action is to read the log, you must 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 defined with the DATA CAPTURE CHANGES clause, and a db2ReadLogInfoStruct structure with the current active log information. If the requested action is a query of the database log (indicated by specifying the value DB2READLOG_QUERY), the API returns a 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 that are 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 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.

This API reads data from the DB2 logs. Label-based access control (LBAC) is not enforced on such logs. Thus, an application that calls this API can gain access to table data if the caller has sufficient authority to call the API and is able to understand the log records format.

The db2ReadLog API works on the current database connection. If multiple database connections are created with the same process, then use the concurrent access APIs to manage the multiple contexts.

Calling the db2ReadLog API from an application can result in an error when the application disconnects from the database if a commit or rollback is not performed before the disconnect:

- A CLI0116E error might be generated if the db2ReadLog API is called from a CLI application.
- A SQL0428N error might be generated if the db2ReadLog API is called from an embedded SQL application written in C.

Workaround 1: For non-embedded SQL applications, set autocommit mode on before calling the db2ReadLog API.

Workaround 2: Issue a COMMIT or ROLLBACK statement after calling the db2ReadLog API and before disconnecting from the database.

Related reference:

- “db2ReadLogNoConnInit - Initialize reading the database logs without a database connection” on page 262
- “db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection” on page 264
- Appendix B, “DB2 log records,” on page 725
- “SQLCA ” on page 526
- “db2ReadLogNoConn - Read the database logs without a database connection” on page 259

Related samples:

db2ReadLog - Extracts log records

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

db2ReadLogNoConn - Read the database logs without a database connection

Extracts log records from the DB2 database logs and queries the Log Manager for current log state information. Prior to using this API, call the `db2ReadLogNoConnInit` API to allocate the memory that is passed as an input parameter to this API. After calling this API, call the `db2ReadLogNoConnTerm` API to deallocate the memory.

Authorization:

None

Required connection:

None

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
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;
```

db2ReadLogNoConn API parameters:

versionNumber

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.

db2ReadLogNoConn - Read the database logs without a database connection

db2ReadLogNoConnStruct data structure parameters:

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.

db2ReadLogNoConnInfoStruct data structure parameters:

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.

db2ReadLogNoConn - Read the database logs without a database connection

logBytesWritten

Number of bytes written to the log buffer field, poLogBuffer.

lastLogFullyRead

Number indicating the last log file that was read to completion.

currentTimeValue

Reserved for future use.

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 log records returned by db2ReadLogNoConn can be found in the DB2 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.

This API reads data from the DB2 logs. Label-based access control (LBAC) is not enforced on such logs. Thus, an application that calls this API can potentially gain access to table data if the caller has sufficient authority to call the API and is able to understand the log records format.

Related reference:

- “db2ReadLogNoConnInit - Initialize reading the database logs without a database connection” on page 262
- “db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection” on page 264
- Appendix B, “DB2 log records,” on page 725
- “SQLCA ” on page 526
- “db2ReadLog - Extracts log records” on page 254

db2ReadLogNoConnInit - Initialize reading the database logs without a database connection

Allocates the memory to be used by db2ReadLogNoConn in order to extract log records from the DB2 database logs and query the Log Manager for current log state information.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
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;
```

db2ReadLogNoConnInit API parameters:

versionNumber

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.

db2ReadLogNoConnInitStruct data structure parameters:

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.

db2ReadLogNoConnInit - Initialize reading the database logs without a database connection

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:

- “db2ReadLogNoConn - Read the database logs without a database connection” on page 259
- “db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection” on page 264
- Appendix B, “DB2 log records,” on page 725
- “SQLCA ” on page 526

db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection

Deallocates the memory used by the db2ReadLogNoConn API, originally initialized by the db2ReadLogNoConnInit API.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2ReadLogNoConnTerm (
    db2UInt32 versionNumber,
    void * pDB2ReadLogNoConnTermStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
    char                               **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
```

db2ReadLogNoConnTerm API parameters:

versionNumber

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.

db2ReadLogNoConnTermStruct data structure parameters:

poReadLogMemPtr

Output. Pointer to the block of memory allocated in the initialization call. This pointer will be freed and set to NULL.

Related reference:

- “db2ReadLogNoConn - Read the database logs without a database connection” on page 259
- “db2ReadLogNoConnInit - Initialize reading the database logs without a database connection” on page 262
- Appendix B, “DB2 log records,” on page 725
- “SQLCA ” on page 526

db2Recover - Restore and roll forward a database

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. If no database partition servers are specified, it affects all database partition servers that are listed in the db2nodes.cfg file. If a point in time is specified, the API affects all database partitions.

Authorization:

To recover an existing database, one of the following:

- sysadm
- sysctrl
- sysmaint

To recover to a new database, one of the following:

- sysadm
- sysctrl

Required connection:

To recover an existing database, a database connection is required. This API automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. Instance and database, to recover to a new database. The instance attachment is required to create the database.

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Recover (
    db2UInt32 versionNumber,
    void * pDB2RecovStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RecoverStruct
{
    char *piSourceDBAlias;
    char *piUsername;
    char *piPassword;
    db2UInt32 iRecoverCallerAction;
    db2UInt32 iOptions;
    sqlint32 *poNumReplies;
    struct sqlurf_info *poNodeInfo;
    char *piStopTime;
    char *piOverflowLogPath;
    db2UInt32 iNumChngLgOvrflw;
    struct sqlurf_newlogpath *piChngLogOvrflw;
    db2int32 iAllNodeFlag;
    db2int32 iNumNodes;
    SQL_PDB_NODE_TYPE *piNodeList;
    db2int32 iNumNodeInfo;
    char *piHistoryFile;
```

db2Recover - Restore and roll forward a database

```
    db2UInt32 iNumChngHistoryFile;
    struct sqlu_histFile *piChngHistoryFile;
    char *piComprLibrary;
    void *piComprOptions;
    db2UInt32 iComprOptionsSize;
} db2RecoverStruct;

SQL_STRUCTURE sqlu_histFile
{
    SQL_PDB_NODE_TYPE nodeNum;
    unsigned short filenameLen;
    char filename[SQL_FILENAME_SZ+1];
};

SQL_API_RC SQL_API_FN
db2gRecover (
    db2UInt32 versionNumber,
    void * pDB2gRecoverStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRecoverStruct
{
    char *piSourceDBAlias;
    db2UInt32 iSourceDBAliasLen;
    char *piUserName;
    db2UInt32 iUserNameLen;
    char *piPassword;
    db2UInt32 iPasswordLen;
    db2UInt32 iRecoverCallerAction;
    db2UInt32 iOptions;
    sqlint32 *poNumReplies;
    struct sqlurf_info *poNodeInfo;
    char *piStopTime;
    db2UInt32 iStopTimeLen;
    char *piOverflowLogPath;
    db2UInt32 iOverflowLogPathLen;
    db2UInt32 iNumChngLgOvrflw;
    struct sqlurf_newlogpath *piChngLogOvrflw;
    db2int32 iAllNodeFlag;
    db2int32 iNumNodes;
    SQL_PDB_NODE_TYPE *piNodeList;
    db2int32 iNumNodeInfo;
    char *piHistoryFile;
    db2UInt32 iHistoryFileLen;
    db2UInt32 iNumChngHistoryFile;
    struct sqlu_histFile *piChngHistoryFile;
    char *piComprLibrary;
    db2UInt32 iComprLibraryLen;
    void *piComprOptions;
    db2UInt32 iComprOptionsSize;
} db2gRecoverStruct;
```

db2Recover API parameters:

versionNumber

Input. Specifies the version and release level of the structure passed as the second parameter pDB2RecoverStruct.

pDB2RecoverStruct

Input. A pointer to the db2RecoverStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2RecoverStruct data structure parameters:

db2Recover - Restore and roll forward a database

piSourceDBAlias

Input. A string containing the database alias of the database to be recovered.

piUserName

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

piPassword

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

iRecoverCallerAction

Input. Valid values are:

DB2RECOVER

Starts the recover operation. Specifies that the recover will run unattended, and that scenarios that 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 recover have been mounted, and utility prompts are not desired.

DB2RECOVER_RESTART

Allows the user to ignore a prior recover and start over from the beginning.

DB2RECOVER_CONTINUE

Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2RECOVER_LOADREC_TERM

Terminate all devices being used by load recovery.

DB2RECOVER_DEVICE_TERM

Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2RECOVER_PARM_CHK_ONLY

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

DB2RECOVER_DEVICE_TERMINATE

Removes a particular device from the list of devices used by the recover operation. When a particular device has exhausted its input, recover will return a warning to the caller. Call the recover utility again with this caller action to remove the device that generated the warning from the list of devices being used.

iOptions

Input. Valid values are:

- DB2RECOVER_EMPTY_FLAG

No flags specified.

- DB2RECOVER_LOCAL_TIME

Indicates that the value specified for the stop time by piStopTime is in local time, not GMT. This is the default setting.

db2Recover - Restore and roll forward a database

- DB2RECOVER_GMT_TIME

This flag indicates that the value specified for the stop time by piStopTime is in GMT (Greenwich Mean Time).

poNumReplies

Output. The number of replies received.

poNodeInfo

Output. Database partition reply information.

piStopTime

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 DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_) caller actions.

piOverflowLogPath

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 location specified by the logpath configuration parameter before they can be used by this utility. This can be a problem if the user does not have sufficient space in the log path. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the log path, and then in the overflow log path. The log files needed for table space rollforward recovery can be brought into either the log path or the overflow log path. If the caller does not specify an overflow log path, the default value is the log path.

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 database partition. In a single-partition database environment, the overflow log path can be relative if the server is local.

iNumChngLgOvrflw

Input. 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.

piChngLogOvrflw

Input. 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.

iAllNodeFlag

Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST

Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES

Apply to all database partition servers. piNodeList should be NULL. This is the default value.

db2Recover - Restore and roll forward a database

DB2_ALL_EXCEPT

Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY

Apply to the catalog partition only. piNodeList should be NULL.

iNumNodes

Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList

Input. A pointer to an array of database partition server numbers on which to perform the rollforward recovery.

iNumNodeInfo

Input. Defines the size of the output parameter poNodeInfo, 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 the same as the number of database partition servers for which this API is being called.

piHistoryFile

History file.

iNumChngHistoryFile

Number of history files in list.

piChngHistoryFile

List of history files.

piComprLibrary

Input. Indicates the name of the external library to be used to perform decompression of the backup image if the image is compressed. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore will fail.

piComprOptions

Input. Describes a block of binary data that will be passed to the initialization routine in the decompression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize

Input. Represents the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

sqlu_histFile data structure parameters:

nodeNum

Input. Specifies which database partition this entry should be used for.

db2Recover - Restore and roll forward a database

filenameLen

Input. Length in bytes of filename.

filename

Input. Path to the history file for this database partition. The path must end with a slash.

db2gRecoverStruct data structure specific parameters:

iSourceDBAliasLen

Specifies the length in bytes of the piSourceDBAlias parameter.

iUserNameLen

Specifies the length in bytes of the piUsername parameter.

iPasswordLen

Specifies the length in bytes of the piPassword parameter.

iStopTimeLen

Specifies the length in bytes of the piStopTime parameter.

iOverflowLogPathLen

Specifies the length in bytes of the piOverflowLogPath parameter.

iHistoryFileLen

Specifies the length in bytes of the piHistoryFile parameter.

iComprLibraryLen

Input. Specifies the length in bytes of the name of the library specified in the piComprLibrary parameter. Set to zero if no library name is given.

Related tasks:

- “Using recover” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “SQLCA ” on page 526
- “RECOVER DATABASE command” in *Command Reference*
- “db2Rollforward - Roll forward a database” on page 296
- “db2Backup - Back up a database or table space” on page 41
- “db2Restore - Restore a database or table space” on page 284

db2Reorg - Reorganize an index or a table

Reorganizes 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

API and data structure syntax:

```
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;
    char *pLongTempSpace;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
    char *pTableName;
    char *pIndexName;
} db2ReorgIndexesAll;

SQL_API_RC SQL_API_FN
```

db2Reorg - Reorganize an index or a table

```
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;
    db2Uint32 longTempSpaceLen;
    char *pLongTempSpace;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
    db2Uint32 tableNameLen;
    char *pTableName;
    db2Uint32 indexNameLen;
    char *pIndexName;
} db2gReorgIndexesAll;
```

db2Reorg 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.

db2ReorgStruct data structure parameters:

reorgType

Input. Specifies the type of reorganization. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2REORG_OBJ_TABLE_OFFLINE

Reorganize the table offline.

DB2REORG_OBJ_TABLE_INPLACE

Reorganize the table inplace.

DB2REORG_OBJ_INDEXESALL

Reorganize all indexes.

DB2REORG_OBJ_INDEX

Reorganize one index.

reorgFlags

Input. Reorganization options. Valid values (defined in db2ApiDf header file, located in the include directory) 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 or DB2REORG_OBJ_INDEX are specified as the reorgType.

DB2REORG_CLEANUP_ALL

Clean up the committed pseudo deleted keys and committed

db2Reorg - Reorganize an index or a table

pseudo empty pages, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are 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 or DB2REORG_OBJ_INDEX are specified as the reorgType.

DB2REORG_CONVERT_NONE

No conversion is required, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the reorgType.

DB2REORG_CONVERT

Convert to type 2 index, used when DB2REORG_OBJ_INDEXESALL is specified as the reorgType.

DB2REORG_RESETDICTIONARY

If the COMPRESS attribute for the table is YES then a new row compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary. If the COMPRESS attribute for the table is NO and the table does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in non-compressed format. This parameter is only supported for the DB2REORG_OBJ_TABLE_OFFLINE reorgType.

DB2REORG_KEEPMONITOR

If the COMPRESS attribute for the table is YES and a dictionary exists, it is kept. If the COMPRESS attribute for the table is YES and a dictionary does not exist, one is built, as the option defaults to DB2REORG_RESETDICTIONARY in that case. All rows processed by reorganization are subject to compression. If the COMPRESS attribute for the table is NO, the dictionary will be retained (if one existed), and all rows in the newly reorganized table will be in non-compressed format. This parameter is only supported for the DB2REORG_OBJ_TABLE_OFFLINE reorgType.

nodeListFlag

Input. Specifies which nodes to reorganize. Valid values (defined in db2ApiDf header file, located in the include directory) 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.

db2ReorgObject union parameters:

tableStruct

Specifies the options for a table reorganization.

indexesAllStruct

Specifies the options for an index reorganization.

db2ReorgTable data structure parameters:

pTableName

Input. Specifies the name of the table to reorganize.

pOrderByIndex

Input. Specifies the index to order the table by.

pSysTempSpace

Input. Specifies the system temporary table space where temporary objects are created. The REORG command may expand rows in cases where a column is added to a table (i.e. from ALTER TABLE ADD COLUMN) and the rows were inserted before the column was added. For a non-partitioned table, this parameter must specify a table space with enough room to create the new table object. A partitioned table is reorganized a single data partition at a time. In this case, there must be enough free space in the table space to hold the largest data partition of the table.

If this parameter is not specified for a non-partitioned table the table space the table resides in is used. If this parameter is not specified for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

pLongTempSpace

Input. Specifies the temporary table space to create long objects (LONG VARCHAR and LOB columns) in during table reorganization. If the pSysTempSpace parameter is not specified, this parameter is ignored. If this parameter is not specified, but the pSysTempSpace parameter is specified, then DB2 will create the long data objects in the table space specified by the pSysTempSpace parameter, unless the page sizes differ.

When page sizes differ, if pSysTempSpace is specified, but this parameter is not, DB2 will attempt to find an existing table space with a matching page size to create the long objects in.

db2ReorgIndexesAll data structure parameters:

pTableName

Input. Specifies the name of the table for index reorganization. If DB2REORG_OBJ_INDEX is specified as the reorgType, the pTableName parameter is not required and can be NULL. However, if the pTableName parameter is specified, it must be the table on which the index is defined.

pIndexName

Input. Specifies the name of the index to reorganize. This parameter is used only when the reorgType parameter is set to a value of DB2REORG_OBJ_INDEX otherwise set pIndexName parameter to NULL.

db2gReorgTable data structure specific parameters:

db2Reorg - Reorganize an index or a table

tableNameLen

Input. Specifies the length in bytes of pTableName.

orderByIndexLen

Input. Specifies the length in byte of pOrderByIndex.

sysTempSpaceLen

Input. Specifies the length in bytes of pSysTempSpace.

longTempSpaceLen

Input. Specifies the length of the name stored in the pLongTempSpace parameter.

db2gReorgIndexesAll data structure specific parameters:

tableNameLen

Input. Specifies the length in bytes of pTableName.

indexNameLen

Input. Specifies the length in bytes of the pIndexName parameter.

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 during reorg processing. After reorganizing a table or its indexes, use db2Runstats to update the statistics and sqlarbind to rebind the packages that use this table.
- If the table data is distributed onto several nodes and the reorganization fails on any of the affected nodes, then only the failing nodes will have the reorganization rolled back. If table reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.
- For table reorganization, 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 not defined with 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).
- The following table illustrates the default table access chosen based on the type of reorg and table:

Table 22. Default table access chosen based on the type of reorg and table

Type of reorg and applicable flags which can affect the default table access		Access mode chosen for each table type	
reorgType	reorgFlags (if applicable)	Non-partitioned Table	Partitioned Table
DB2REORG_OBJ_TABLE_OFFLINE		DB2REORG_ALLOW_READ	DB2REORG_ALLOW_NONE

Table 22. Default table access chosen based on the type of reorg and table (continued)

Type of reorg and applicable flags which can affect the default table access		Access mode chosen for each table type	
reorgType	reorgFlags (if applicable)	Non-partitioned Table	Partitioned Table
DB2REORG_OBJ_TABLE_INPLACE		DB2REORG_ALLOW_WRITE	N/A
DB2REORG_OBJ_INDEXESALL		DB2REORG_ALLOW_READ	DB2REORG_ALLOW_NONE
DB2REORG_OBJ_INDEXESALL	DB2REORG_CLEANUP_ALL, DB2REORG_CLEANUP_PAGES	DB2REORG_ALLOW_READ	DB2REORG_ALLOW_READ
DB2REORG_OBJ_INDEX		N/A	DB2REORG_ALLOW_READ
DB2REORG_OBJ_INDEX	DB2REORG_CLEANUP_ALL, DB2REORG_CLEANUP_PAGES	N/A	DB2REORG_ALLOW_READ

N/A: Not applicable at this time since it is not supported.

Note: Some access modes may not be supported on certain types of tables or indexes. In these cases and where possible, the least restrictive access mode is used. (The most restrictive access mode being DB2REORG_ALLOW_NONE, followed by DB2REORG_ALLOW_READ, and then DB2REORG_ALLOW_WRITE, which is the least restrictive). As support for existing table or index types change, or new table or index types are provided, the default can change from a more restrictive access mode to a less restrictive mode. The least restrictive mode chosen for the default will not go beyond DB2REORG_ALLOW_READ when the reorgType is not DB2REORG_OBJ_TABLE_INPLACE. The default access mode is chosen when none of the DB2REORG_ALLOW_NONE, DB2REORG_ALLOW_READ, or DB2REORG_ALLOW_WRITE flags are specified.

- 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 index(es) 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 you about the change in the access mode. For an index reorganization of a partitioned table, any indexes that need to be rebuilt are rebuilt offline and then the index you specified is reorganized, assuming it was not already rebuilt. This reorganization uses the access mode specified by you. A message will be written to the administration notification log and the diagnostics log to warn you that the indexes are being rebuilt offline.
- For non-inplace table reorganization, if neither DB2REORG_RESETDICTIONARY or DB2REORG_KEEPPDICTIONARY is specified, the default is DB2REORG_KEEPPDICTIONARY.
- If an index reorganization with no access fails, some or all indexes will not be available and will 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 concepts:

db2Reorg - Reorganize an index or a table

- “Table reorganization” in *Performance Guide*

Related reference:

- “REORG INDEXES/TABLE command” in *Command Reference*
- “REORGCHK command” in *Command Reference*
- “ADMIN_CMD procedure – Run administrative commands” in *Administrative SQL Routines and Views*
- “db2Runstats - Update statistics for tables and indexes” on page 306
- “sqlarbnd - Rebind package” on page 358
- “SQLCA ” on page 526

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 the alert configuration of health indicators

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2ResetAlertCfg (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ResetAlertCfgData
{
    db2UInt32 iObjType;
    char *piObjName;
    char *piDbName;
    db2UInt32 iIndicatorID;
} db2ResetAlertCfgData;
```

db2ResetAlertCfg 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.

db2ResetAlertCfgData data structure parameters:

iObjType

Input. Specifies the type of object for which configuration should be reset. Valid values (defined in db2ApiDf header file, located in the include directory) are:

- DB2ALERTCFG_OBJTYPE_DBM
- DB2ALERTCFG_OBJTYPE_DATABASES
- DB2ALERTCFG_OBJTYPE_TABLESPACES

db2ResetAlertCfg - Reset the alert configuration of health indicators

- 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-container-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.

iIndicatorID

Input. The health indicator for which the configuration resets are to apply.

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 tasks:

- “Configuring health indicators using a client application” in *System Monitor Guide and Reference*

Related reference:

- “RESET ALERT CONFIGURATION command” in *Command Reference*
- “db2GetAlertCfg - Get the alert configuration settings for the health indicators” on page 95
- “db2UpdateAlertCfg - Update the alert configuration settings for health indicators” on page 328
- “SQLCA ” on page 526
- “RESET ALERT CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

db2ResetMonitor - Reset the database system monitor data

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
- sysmon

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2ResetMonitor (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ResetMonitorData
{
    db2UInt32 iResetAll;
    char *piDbAlias;
    db2UInt32 iVersion;
    db2int32 iNodeNumber;
} db2ResetMonitorData;

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;
```

db2ResetMonitor - Reset the database system monitor data

db2ResetMonitor 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.

db2ResetMonitorData data structure parameters:

iResetAll

Input. The reset flag.

piDbAlias

Input. A pointer to the database alias.

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 the value, SQLM_CURRENT_NODE, must be used.

db2gResetMonitorData data structure specific parameters:

iDbAliasLength

Input. Specifies the length in bytes of the piDbAlias parameter.

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

db2ResetMonitor - Reset the database system monitor data

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:

- “RESET MONITOR command” in *Command Reference*
- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API” on page 117
- “db2MonitorSwitches - Get or update the monitor switch settings” on page 246
- “SQLCA ” on page 526

db2Restore - Restore a database or table space

Recreates a damaged or corrupted database that has been backed up using the db2Backup API. 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, or restore table spaces from within a database backup image.

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Restore (
    db2UInt32 versionNumber,
    void * pDB2RestoreStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RestoreStruct
```

db2Restore - Restore a database or table space

```
{
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;
char *piComprLibrary;
void *piComprOptions;
db2Uint32 iComprOptionsSize;
char *piLogTarget;
struct db2StoragePathsStruct *piStoragePaths;
char *piRedirectScript;
} db2RestoreStruct;

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

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

typedef SQL_STRUCTURE db2StoragePathsStruct
{
char **storagePaths;
db2Uint32 numStoragePaths;
} db2StoragePathsStruct;

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;
```

db2Restore - Restore a database or table space

```
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;
char *piComprLibrary;
db2Uint32 iComprLibraryLen;
void *piComprOptions;
db2Uint32 iComprOptionsSize;
char *piLogTarget;
db2Uint32 iLogTargetLen;
struct db2gStoragePathsStruct *piStoragePaths;
char *piRedirectScript;
db2Uint32 iRedirectScriptLen;
} db2gRestoreStruct;

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 db2gStoragePathsStruct
{
    struct db2Char *storagePaths;
    db2Uint32 numStoragePaths;
} db2gStoragePathsStruct;

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

db2Restore 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.

db2RestoreStruct data structure parameters:

piSourceDBAlias

Input. A string containing the database alias of the source database backup image.

piTargetDBAlias

Input. A string containing the target database alias. If this parameter is null, the value of the piSourceDBAlias parameter will be used.

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.

piTimestamp

Input. A string representing the time stamp of the backup image. This field is optional if there is only one backup image in the source specified.

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 a restore (as a result of a fast reconcile) will be reported.

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. For rebuild cases, this can be an include list or exclude list of table spaces used to rebuild your database. See the DB2TablespaceStruct structure. The following restrictions apply:

- The database must be recoverable (for non-rebuild cases only); 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.
- In the case of rebuild, the list must be given for 3 of the 5 rebuild types: DB2RESTORE_ALL_TBSP_IN_DB_EXC, DB2RESTORE_ALL_TBSP_IN_IMG_EXC and DB2RESTORE_ALL_TBSP_IN_LIST

piMediaList

Input. Source media for the backup image. The information provided depends on the value of the locationType parameter. The valid values for locationType parameter (defined in sqlutil header file, located in the include directory) are:

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

db2Restore - Restore a database or table space

- `SQLU_XBSA_MEDIA` - XBSA interface. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.
- `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`.

piPassword

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

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.

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 in bytes of the `piVendorOptions` parameter, which cannot exceed 65535 bytes.

iParallelism

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

iBufferSize

Input. Backup buffer size in 4 KB allocation units (pages). Minimum is 8 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` header file, located in the include directory) 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 - Restore a database or table space

- 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 a successful return of this call, it is expected that the user will issue another call to this API with the iCallerAction parameter set to the value DB2RESTORE_CONTINUE to continue with the restore.
- 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 header file, located in the include directory) 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_COMPR_LIB - Indicates that the compression library is to be restored. This option cannot be used simultaneously with any other type of restore process. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.
- DB2RESTORE_LOGS - Specifies that only the set of log files contained in the backup image are to be restored. If the backup image does not include log files, the restore operation will fail. If this option is specified, the piLogTarget parameter must also be specified.
- DB2RESTORE_INCREMENTAL - Perform a manual cumulative restore operation.
- DB2RESTORE_AUTOMATIC - Perform an automatic cumulative (incremental) restore operation. Must be specified with DB2RESTORE_INCREMENTAL.

db2Restore - Restore a database or table space

- 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 data type 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, the db2Rollforward API must be called before the database can be used.
- DB2RESTORE_GENERATE_SCRIPT - Create a script, that can be used to perform a redirected restore. piRedirectScript must contain a valid file name. The iCallerAction need to be either DB2RESTORE_RESTORE_STORDEF or DB2RESTORE_STORDEF_NOINTERRUPT.

The following values should be used for rebuild operations only:

- DB2RESTORE_ALL_TBSP_IN_DB - Restores the database with all the table spaces known to the database at the time of the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_DB_EXC - Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG - Restores the database with only the table spaces in the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG_EXC - Restores the database with only the table spaces in the image being restored except for those specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_LIST - Restores the database with only the table spaces specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.

NOTE: If the backup image is of a recoverable database then WITHOUT ROLLING FORWARD (DB2RESTORE_NOROLLFWD) cannot be specified with any of the above rebuild actions.

piComprLibrary

Input. Indicates the name of the external library to use to decompress the backup image if the image is compressed. The name must be a fully-qualified path that refers to a file on the server. If the value is a null pointer or a pointer to an empty string, the DB2 database system attempts to use the library stored in the image. If the backup is not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore operation will fail.

piComprOptions

Input. This API parameter describes a block of binary data that will be

db2Restore - Restore a database or table space

passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte-reversal or code-page conversion must be handled by the compression library. If the first character of the data block is '@', the remainder of the data is interpreted as the name of a file residing on the server. The DB2 database system then replaces the contents of the piComprOptions and iComprOptionsSize parameters with the contents and size of this file and passes these new values to the initialization routine.

iComprOptionsSize

Input. A four-byte unsigned integer that represents the size of the block of data passed as piComprOptions. The iComprOptionsSize parameter should be zero if and only if the piComprOptions value is a null pointer.

piLogTarget

Input. Specifies the absolute path of a directory on the database server that must be used as the target directory for extracting log files from a backup image. If this parameter is specified, any log files included in the backup image are extracted into the target directory. If this parameter is not specified, log files included in the backup image are not extracted. To extract only the log files from the backup image, DB2RESTORE_LOGS value should be passed to the iOptions parameter.

piStoragePaths

Input. A structure containing fields that describe a list of storage paths used for automatic storage. Set this to NULL if automatic storage is not enabled for the database.

piRedirectScript

Input. The file name for the redirect restore script that will be created on client side. The file name can be specified relative or absolute. The iOptions field need to have the DB2RESTORE_GENERATE_SCRIPT bit set.

db2TablespaceStruct data structure specific parameters:

tablespaces

Input. 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

Input. Number of entries in the tablespaces parameter.

db2MediaListStruct data structure parameters:

locations

Input. 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

Input. The number of entries in the locations parameter.

locationType

Input. A character indicating the media type. Valid values (defined in sqlutil header file, located in the include directory.) are:

- SQLU_LOCAL_MEDIA - Local devices (tapes, disks, diskettes, or named pipes).
- SQLI_XBSA_MEDIA - XBSA interface.
- SQLU_TSM_MEDIA - Tivoli Storage Manager.

db2Restore - Restore a database or table space

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

db2StoragePathsStruct data structure parameters:

storagePaths

Input. An array of strings containing fully qualified names of storage paths on the server that will be used for automatic storage table spaces. In a multi-partition database the same storage paths are used on all database partitions. If a multi-partiton database is being restored with new storage paths, then the catalog partition must be restored before any other database partitions are restored.

numStoragePaths

Input. The number of storage paths in the storagePaths parameter of the db2StoragePathsStruct structure.

db2gRestoreStruct data structure specific parameters:

iSourceDBAliasLen

Input. Specifies the length in bytes of the piSourceDBAlias parameter.

iTargetDBAliasLen

Input. Specifies the length in bytes of the piTargetDBAlias parameter.

iApplicationIdLen

Input. Specifies the length in bytes of the poApplicatoinId parameter. Should be equal to SQLU_APPLID_LEN + 1. The constant SQLU_APPLID_LEN is defined in sqlutil header file that is located in the include directory.

iTimestampLen

Input. Specifies the length in bytes of the piTimestamp parameter.

iTargetDBPathLen

Input. Specifies the length in bytes of the piTargetDBPath parameter.

iReportFileLen

Input. Specifies the length in bytes of the piReportFile parameter.

iUsernameLen

Input. Specifies the length in bytes of the piUsername parameter. Set to zero if no user name is provided.

iPasswordLen

Input. Specifies the length in bytes of the piPassword parameter. Set to zero if no password is provided.

iNewLogPathLen

Input. Specifies the length in bytes of the piNewLogPath parameter.

iLogTargetLen

Input. Specifies the length in bytes of the piLogTarget parameter.

iRedirectScriptLen

Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in piRedirectScript. Set to zero if no script name is given.

db2Char data structure parameters:

pioData

A pointer to a character data buffer. If NULL, no data will be returned.

iLength

Input. The size of the pioData buffer.

oLength

Output. The number of valid characters of data in the pioData buffer.

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. In this case, if the file is replaced, a warning message is returned.
- The database or table space must have been backed up using the db2Backup API.
- If the caller action value is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action value 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 value 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 value 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 value 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 value to DB2RESTORE_PARM_CHK.
- Set caller action value to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with the sqlbstsc API.
- 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.

db2Restore - Restore a database or table space

- If the restore type specifies that the history file in 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.
- To restore log files from a backup image that contains them, the LOGTARGET option must be specified, assuming a fully qualified and valid path exists on the DB2 server. If those conditions are satisfied, the restore utility writes the log files from the image to the target path. If LOGTARGET is specified during a restoration of a backup image that does not include logs, the restore operation returns an error before attempting to restore any table space data. A restore operation also fails with an error if an invalid or read-only LOGTARGET path is specified.
- If any log files exist in the LOGTARGET path at the time the RESTORE command is issued, a warning prompt is returned to user. This warning is not returned if WITHOUT PROMPTING is specified.
- During a restore operation in which a LOGTARGET is specified, if any log file cannot be extracted, the restore operation fails and returns an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation fails and an error is returned. The restore utility does not overwrite existing log files in the LOGTARGET directory.
- You can restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path results in an error. If any problem occurs while restoring log files in this mode the restore operation terminates immediately and an error is returned.
- During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved from the backup image. Any logs that are included in intermediate images that are referenced during the incremental restore process are not extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should be specified only with the final restore command.
- If a backup is compressed, the DB2 database system detects this state and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. If a library is not specified on the db2Restore API, the library stored in the backup image is used. And if there is no library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is being restored from a backup image (either explicitly by specifying the DB2RESTORE_COMPR_LIB restore type or implicitly by performing a normal restoration of a compressed backup), the restore operation must be done on the same platform and operating system that the

db2Restore - Restore a database or table space

backup was taken on. If the platforms are different, the restore operation will fail, even when the DB2 database system normally supports cross-platform restore operations involving the two systems.

- If restoring a database that is enabled for automatic storage, the storage paths associated with the database can be redefined or they can remain as they were previously. To keep the storage path definitions as is, do not provide any storage paths as part of the restore operation. Otherwise, specify a new set of storage paths to associate with the database. Automatic storage table spaces will be automatically redirected to the new storage paths during the restore operation.

Related tasks:

- “Using restore” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “sqlmgdb - Migrate previous version of DB2 database to current version” on page 455
- “SQLCA ” on page 526
- “RESTORE DATABASE command” in *Command Reference*
- “db2Rollforward - Roll forward a database” on page 296
- “db2Backup - Back up a database or table space” on page 41
- “db2Recover - Restore and roll forward a database” on page 265

Related samples:

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

db2Rollforward - Roll forward a 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 the `logarchmeth1` database configuration parameter or the `logarchmeth2` database configuration parameter must be set to a value other than OFF) before the database can be recovered with rollforward 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.

For partitioned tables, you are also required to roll forward related table spaces to the same point in time. Related table spaces contain attached, detached, and dropped data partitions or indexes of a table. Rollforward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

None. This API establishes a database connection.

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Rollforward (
    db2UInt32 versionNumber,
    void * pDB2RollforwardStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
    struct db2RfwdInputStruct *piRfwdInput;
    struct db2RfwdOutputStruct *poRfwdOutput;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
    sqluint32 iVersion;
    char *piDbAlias;
```

```

db2UInt32 iCallerAction;
char *piStopTime;
char *piUserName;
char *piPassword;
char *piOverflowLogPath;
db2UInt32 iNumChngLgOvrflw;
struct sqlurf_newlogpath *piChngLogOvrflw;
db2UInt32 iConnectMode;
struct sqlu_tablespace_bkrst_list *piTablespaceList;
db2int32 iAllNodeFlag;
db2int32 iNumNodes;
SQL_PDB_NODE_TYPE *piNodeList;
db2int32 iNumNodeInfo;
char *piDroppedTblID;
char *piExportDir;
db2UInt32 iRollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
    char *poApplicationId;
    sqlint32 *poNumReplies;
    struct sqlurf_info *poNodeInfo;
    db2UInt32 oRollforwardFlags;
} db2RfwdOutputStruct;

SQL_STRUCTURE sqlurf_newlogpath
{
    SQL_PDB_NODE_TYPE nodenum;
    unsigned short pathlen;
    char logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
    sqlint32 num_entry;
    struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;

typedef SQL_STRUCTURE sqlu_tablespace_entry
{
    sqluint32 reserve_len;
    char tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
    char filler[1];
} sqlu_tablespace_entry;

SQL_STRUCTURE sqlurf_info
{
    SQL_PDB_NODE_TYPE nodenum;
    sqlint32 state;
    unsigned char nextarclog[SQLUM_ARCHIVE_FILE_LEN+1];
    unsigned char firstarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
    unsigned char lastarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
    unsigned char lastcommit[SQLUM_TIMESTAMP_LEN+1];
};

SQL_API_RC SQL_API_FN
db2gRollforward (
    db2UInt32 versionNumber,
    void * pDB2gRollforwardStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
    struct db2RfwdInputStruct *piRfwdInput;
    struct db2RfwdOutputStruct *poRfwdOutput;
} db2gRollforwardStruct;

```

db2Rollforward - Roll forward a database

```
typedef SQL_STRUCTURE db2gRfwdInputStruct
{
    db2UInt32 iDbAliasLen;
    db2UInt32 iStopTimeLen;
    db2UInt32 iUserNameLen;
    db2UInt32 iPasswordLen;
    db2UInt32 iOvrflwLogPathLen;
    db2UInt32 iDroppedTblIDLen;
    db2UInt32 iExportDirLen;
    sqluint32 iVersion;
    char *piDbAlias;
    db2UInt32 iCallerAction;
    char *piStopTime;
    char *piUserName;
    char *piPassword;
    char *piOverflowLogPath;
    db2UInt32 iNumChngLgOvrflw;
    struct sqlurf_newlogpath *piChngLogOvrflw;
    db2UInt32 iConnectMode;
    struct sqlu_tablespace_bkrst_list *piTablespaceList;
    db2int32 iAllNodeFlag;
    db2int32 iNumNodes;
    SQL_PDB_NODE_TYPE *piNodeList;
    db2int32 iNumNodeInfo;
    char *piDroppedTblID;
    char *piExportDir;
    db2UInt32 iRollforwardFlags;
} db2gRfwdInputStruct;
```

db2Rollforward 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.

db2RollforwardStruct data structure parameters:

piRfwdInput

Input. A pointer to the db2RfwdInputStruct structure.

poRfwdOutput

Output. A pointer to the db2RfwdOutputStruct structure.

db2RfwdInputStruct data structure parameters:

iVersion

Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

piDbAlias

Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

iCallerAction

Input. Specifies action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_ROLLFWD

Rollforward to the point in time specified by the piStopTime parameter. 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.

DB2ROLLFORWARD_STOP

End roll-forward recovery by rolling forward the database using available log files and then rolling it back. Uncommitted transactions are backed out and the rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_RFWD_STOP

Rollforward to the point in time specified by piStopTime, and end roll-forward recovery. The rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_QUERY

Query values for nextarclog, firstarcdel, lastarcdel, and lastcommit. Return database status and a node number.

DB2ROLLFORWARD_PARM_CHECK

Validate parameters without performing the roll forward.

DB2ROLLFORWARD_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:

DB2ROLLFORWARD_LOADREC_CONT

Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2ROLLFORWARD_DEVICE_TERM

Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2ROLLFORWARD_LOAD_REC_TERM

Terminate all devices being used by load recovery.

piStopTime

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 DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_XXX) caller actions.

piUserName

Input. A string containing the user name of the application. Can be NULL.

db2Rollforward - Roll forward a database

piPassword

Input. A string containing the password of the supplied user name (if any). Can be NULL.

piOverflowLogPath

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 database 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.

iNumChngLgOvrflw

Input. 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.

piChngLogOvrflw

Input. 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.

iConnectMode

Input. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_OFFLINE

Offline roll forward. This value must be specified for database roll-forward recovery.

DB2ROLLFORWARD_ONLINE

Online roll forward.

piTablespaceList

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.

For partitioned tables, point in time (PIT) roll-forward of a table space containing any piece of a partitioned table must also roll forward all of the other table spaces in which that table resides to the same point in time. Roll forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached, detached or dropped data partitions, then PIT roll-forward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it is generally necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory

specified in the piExportDir parameter. It is possible to roll forward all table spaces in one command, or do repeated roll-forward operations for subsets of the table spaces involved. A warning will be written to the notify log if the db2Rollforward API did not specify the full set of the table spaces necessary to recover all the data for the table. A warning will be returned to the user with full details of all partitions not recovered on the command found in the administration notification log.

Allowing the roll forward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

iAllNodeFlag

Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST

Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES

Apply to all database partition servers. This is the default value. The piNodeList parameter must be set to NULL, if this value is used.

DB2_ALL_EXCEPT

Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY

Apply to the catalog partition only. The piNodeList parameter must be set to NULL, if this value is used.

iNumNodes

Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList

Input. A pointer to an array of database partition server numbers on which to perform the roll-forward recovery.

iNumNodeInfo

Input. Defines the size of the output parameter poNodeInfo, 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 the same as the number of database partition servers for which this API is being called.

piDroppedTblID

Input. A string containing the ID of the dropped table whose recovery is being attempted. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward command.

piExportDir

Input. The name of the directory into which the dropped table data will be exported.

db2Rollforward - Roll forward a database

iRollforwardFlags

Input. Specifies the rollforward flags. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_EMPTY_FLAG

No flags specified.

DB2ROLLFORWARD_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.

DB2ROLLFORWARD_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.

db2RfwdOutputStruct data structure parameters:

poApplicationId

Output. The application ID.

poNumReplies

Output. The number of replies received.

poNodeInfo

Output. Database partition reply information.

oRollforwardFlags

Output. Rollforward output flags. Valid values are:

DB2ROLLFORWARD_OUT_LOCAL_TIME

Indicates to user that the last committed transaction timestamp is displayed in local time rather than UTC. Local time is based on the server's local time, not on the client's. In a partitioned database environment, local time is based on the catalog partition's local time.

sqlurf_newlogpath data structure parameters:

nodenum

Input. The number of the database partition that this structure details.

pathlen

Input. The total length of the logpath field.

logpath

Input. A fully qualified path to be used for a specific node for the rollforward operation.

sqlu_tablespace_bkrst_list data structure parameters:

num_entry

Input. The number of structures contained in the list pointed to by the tablespace parameter.

tablespace

Input. A pointer to a list of sqlu_tablespace_entry structures.

sqlu_tablespace_entry data structure parameters:**reserve_len**

Input. Specifies the length in bytes of the tablespace_entry parameter.

tablespace_entry

Input. The name of the table space to rollforward.

filler Filler used for proper alignment of data structure in memory.

sqlurf_info data structure parameters:**nodenum**

Output. The number of the database partition that this structure contains information for.

state Output. The current state of the database or tablespaces that were included in the rollforward on a database partition.

nextarclog

Output. If the rollforward has completed, this field will be empty. If the rollforward has not yet completed, this will be the name of the next log file which will be processed for the rollforward.

firstarcdel

Output. The first log file replayed by the rollforward.

lastarcdel

Output. The last log file replayed by the rollforward.

lastcommit

Output. The time of the last committed transaction.

db2gRfwdInputStruct data structure specific parameters:**iDbAliasLen**

Input. Specifies the length in bytes of the database alias.

iStopTimeLen

Input. Specifies the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

iUserNameLen

Input. Specifies the length in bytes of the user name. Set to zero if no user name is provided.

iPasswordLen

Input. Specifies the length in bytes of the password. Set to zero if no password is provided.

iOverflowLogPathLen

Input. Specifies the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

db2Rollforward - Roll forward a database

iDroppedTblIDLen

Input. Specifies the length in bytes of the dropped table ID (piDroppedTblID parameter). Set to zero if no dropped table ID is provided.

iExportDirLen

Input. Specifies the length in bytes of the dropped table export directory (piExportDir parameter). Set to zero if no dropped table export directory is provided.

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 DB2ROLLFORWARD_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 `lascommit` 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 `piStopTime`, 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 `DB2ROLLFORWARD_STOP` or `DB2ROLLFORWARD_RFWRD_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 `DB2ROLLFORWARD_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 tasks:

- "Using rollforward" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "SQLCA " on page 526
- "ROLLFORWARD DATABASE command" in *Command Reference*
- "db2Backup - Back up a database or table space" on page 41
- "db2Restore - Restore a database or table space" on page 284
- "db2Recover - Restore and roll forward a database" on page 265

Related samples:

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

db2Runstats - Update statistics for tables and indexes

Updates statistics about the characteristics of a table and/or any associated indexes or statistical views. 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.

When used on tables, this utility should be called when a table has had many updates, after reorganizing a table, or after creating a new index.

Statistics are based on the portion of the table that resides 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 portion of the table:

- If the API is called from a database partition that contains a portion of the table, the utility executes at this database partition.
- If the API is called from a database partition that does not contain a portion of the table, the request is sent to the first database partition in the database partition group that contains a portion of the table. The utility then executes at this database partition. When you collect statistics for a statistical view, statistics are collected for all database partitions.

When used on statistical views, this utility should be called when changes to underlying tables have substantially affected the rows returned by a view. These views must have been enabled for use in query optimization using "ALTER VIEW ... ENABLE QUERY OPTIMIZATION."

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:

When used on tables, one of the following:

- sysadm
- sysctrl
- sysmaint
- CONTROL privilege on the table
- LOAD

When used on statistical views, one of the following:

- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the view

db2Runstats - Update statistics for tables and indexes

In addition, the user needs to have the appropriate authority or privilege to access rows from the view. Specifically, for each table, view or nickname referenced in the view definition, the user must have one of the following authorities or privileges:

- sysadm or dbadm
- CONTROL privilege
- SELECT privilege

Required connection:

Database

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2Runstats (
    db2UInt32 versionNumber,
    void * data,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RunstatsData
{
    double iSamplingOption;
    unsigned char *piTablename;
    struct db2ColumnData      **piColumnList;
    struct db2ColumnDistData  **piColumnDistributionList;
    struct db2ColumnGrpData   **piColumnGroupList;
    unsigned char             **piIndexList;
    db2UInt32 iRunstatsFlags;
    db2int16 iNumColumns;
    db2int16 iNumColDist;
    db2int16 iNumColGroups;
    db2int16 iNumIndexes;
    db2int16 iParallelismOption;
    db2int16 iTableDefaultFreqValues;
    db2int16 iTableDefaultQuantiles;
    db2UInt32 iSamplingRepeatable;
    db2UInt32 iUtilImpactPriority;
} 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;

SQL_API_RC SQL_API_FN
```

db2Runstats - Update statistics for tables and indexes

```
db2gRunstats (  
    db2UInt32 versionNumber,  
    void * data,  
    struct sqlca * pSqlca);  
  
typedef SQL_STRUCTURE db2gRunstatsData  
{  
    double iSamplingOption;  
    unsigned char *piTablename;  
    struct db2gColumnData      **piColumnList;  
    struct db2gColumnDistData  **piColumnDistributionList;  
    struct 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;  
    db2UInt32 iSamplingRepeatable;  
    db2UInt32 iUtilImpactPriority;  
} 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;
```

db2Runstats 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.

pSqlca

Output. A pointer to the sqlca structure.

db2RunstatsData data structure parameters:

iSamplingOption

Input. Indicates that statistics are to be collected on a sample of table or

db2Runstats - Update statistics for tables and indexes

view data. `iSamplingOption` represents the size of the sample as a percentage `P`. This value must be a positive number that is less than or equal to 100, but may be between 1 and 0. For example, a value of 0.01 represents one one-hundredth of a percent, such that 1 row in 10 000 would be sampled, on average. A value of 0 or 100 will be treated by DB2 as if sampling was not specified, regardless of whether `DB2RUNSTATS_SAMPLING_SYSTEM` has been specified. A value greater than 100 or less than 0 will be treated by DB2 as an error (SQL1197N). The two possible types of sampling are `BERNOULLI` and `SYSTEM`. The sampling type specification is controlled by the indicated setting of `DB2RUNSTATS_SAMPLING_SYSTEM` in the `iRunstatsFlags`.

piTablename

Input. A pointer to the fully qualified name of the table or statistical view 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.

piColumnGrpList

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`.

db2Runstats - Update statistics for tables and indexes

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.

iRunstatsFlags

Input. A bit mask field used to specify statistics options. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2RUNSTATS_ALL_COLUMNS

Collect statistics on all columns of the table or statistical view. 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 or view 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 cannot be used for statistical views. On tables, it 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. XML type columns are, by definition, not key columns and will not be included for statistics collection when the iRunstatsFlags parameter is set to the value DB2RUNSTATS_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 or statistical view. 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. This option cannot be used for statistical views.

DB2RUNSTATS_EXT_INDEX

Collect detailed index statistics. The option must be specified with

db2Runstats - Update statistics for tables and indexes

either DB2RUNSTATS_ALL_INDEXES or an explicit list of index names (piIndexList and iNumIndexes > 0). This option cannot be used for statistical views.

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. This option cannot be used for statistical views.

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.

DB2RUNSTATS_SAMPLING_SYSTEM

Collect statistics on a percentage of the data pages as specified by the user via the iSamplingOption parameter. SYSTEM sampling considers each page individually, including that page with probability $P/100$ (where P is the value of iSamplingOption) and excluding it with probability $1-P/100$. Thus, if iSamplingOption is the value 10, representing a 10 percent sample, each page would be included with probability 0.1 and be excluded with probability 0.9.

SYSTEM sampling cannot be specified on statistical views. Only BERNOULLI sampling can be used to sample view data.

If DB2RUNSTATS_SAMPLING_SYSTEM is not specified, DB2 will assume that BERNOULLI sampling is to be used as the sampling method. BERNOULLI sampling considers each row individually, including that row with probability $P/100$ (where P is the value of iSamplingOption) and excluding it with probability $1-P/100$.

In both SYSTEM and BERNOULLI sampling, unless the DB2RUNSTATS_SAMPLING_REPEAT flag is specified, each execution of statistics collection will usually yield a different sample of the table or statistical view.

DB2RUNSTATS_SAMPLING_REPEAT

Specifies that a seed has been passed through the iSamplingRepeatable parameter. The iSamplingRepeatable value will be used as the seed to generate the data sample. The iSamplingOption parameter must also be specified to indicate the sampling rate.

DB2RUNSTATS_USE_PROFILE

Collect statistics for a table or statistical view by using a statistics profile already registered in the catalogs of the table or view. If the USE PROFILE option is specified by this flag set in iRunstatsFlags bit mask, all other options in db2RunstatsData will be ignored.

DB2RUNSTATS_SET_PROFILE

Generate and store a profile in the catalogs recording the statistics options specified and collect statistics using those same options.

DB2RUNSTATS_SET_PROFILE_ONLY

Generate and store a profile in the catalogs recording the statistics options specified without actually collecting statistics for the table or view.

db2Runstats - Update statistics for tables and indexes

DB2RUNSTATS_UPDATE_PROFILE

Modify an existing statistics profile in the catalogs and collect statistics using the options from the updated profile.

DB2RUNSTATS_UPDATE_PROFILE_ONLY

Modify an existing statistics profile in the catalogs without actually collecting statistics for the table or view.

DB2RUNSTATS_EXCLUDING_XML

Do not collect statistics on XML type columns. Statistics will still be collected on all specified columns that have non-XML type. This option takes precedence over all other methods that specify XML columns.

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 value is 0.

iTableDefaultFreqValues

Input. Specifies the default number of frequent values to collect for the table or view. 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 or view. 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.

iSamplingRepeatable

Input. A non-negative integer representing the seed to be used in table or view sampling. Passing a negative seed will result in an error (SQL1197N).

The DB2RUNSTATS_SAMPLING_REPEAT flag must be set to use this seed. This option is used in conjunction with the iSamplingOption parameter to generate the same sample of data in subsequent statistics collection. The sample set may still vary between repeatable requests if activity against the table or view resulted in changes to the table or view

db2Runstats - Update statistics for tables and indexes

data since the last time a repeatable request was run. Also, the method by which the sample was obtained (BERNOULLI or SYSTEM) must also be the same to ensure consistent results.

iUtilImpactPriority

Input. Priority for the runstats invocation. Valid values must fall in the range 0-100, with 70 representing unthrottled and 100 representing the highest possible priority. This option cannot be used for statistical views.

db2ColumnData data structure parameters:

piColumnName

Input. Pointer to a string representing a 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.

db2ColumnDistData data structure parameters:

piColumnName

Input. Pointer to a string representing a column name.

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`.

db2ColumnGrpData data structure parameters:

piGroupColumnNames

Input. An array of strings. Each string represents a column name that is part of the column group on which to collect statistics.

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.

db2gRunstatsData data structure specific parameters:

db2Runstats - Update statistics for tables and indexes

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.

iTablenameLen

Input. A value representing the length in bytes of the table or view name.

db2gColumnData data structure specific parameters:

iColumnNameLen

Input. A value representing the length in bytes of the column name.

db2gColumnDistData data structure specific parameters:

iColumnNameLen

Input. A value representing the length in bytes of the column name.

db2gColumnGrpData data structure specific parameters:

piGroupColumnNamesLen

Input. An array of values representing the length in bytes of each of the column names in the column names list.

Usage notes:

Use db2Runstats 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.
- On views whose underlying tables have been modified substantially so as to change the rows that are returned by the view.

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.

If the db2Runstats API is collecting statistics on indexes only then previously collected distribution statistics are retained. Otherwise, the API will drop previously collected distribution statistics. If the db2Runstats API is collecting statistics on XML columns only, then previously collected basic column statistics and distribution statistics are retained. In the case where statistics on some XML columns have been collected previously, the previously collected statistics for an XML column will either be dropped if no statistics on that XML column are collected by the current call to the db2Runstats API, or be replaced if statistics on that XML column are collected by the current call to the db2Runstats API. Otherwise, the API will drop previously collected distribution statistics.

If the iRunstatsFlags parameter is set to the value DB2RUNSTATS_EXCLUDING_XML, statistics will not be collected on XML columns. This value takes precedence over all other methods that specify XML columns.

db2Runstats - Update statistics for tables and indexes

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. Packages that contain queries that can take advantage of statistical views must also be rebound after updating statistics on such views.

When statistics are collected for statistical views, an SQL query is run internally. The EXPLAIN facility can be used to examine the access plan selected for this query to investigate any performance problems with the statistics collection. To save the query access plan in the EXPLAIN tables, set the CURRENT EXPLAIN MODE special register to YES.

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 (FIRSTKEYCARD is a catalog statistics field in SYSCAT.INDEXES and SYSSTAT.INDEXES catalog views) 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 (COLCARD is a catalog statistics field in SYSCAT.COLUMNS and SYSSTAT.COLUMNS catalog views) greater than the table cardinality. A warning will be returned if such an inconsistency is detected.

Related reference:

- “REORGCHK command” in *Command Reference*
- “RUNSTATS command” in *Command Reference*
- “ADMIN_CMD procedure – Run administrative commands” in *Administrative SQL Routines and Views*
- “db2Reorg - Reorganize an index or a table” on page 271
- “sqlabndx - Bind application program to create a package” on page 350
- “SQLCA ” on page 526
- “RUNSTATS command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

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++)”

db2SelectDB2Copy - Select the DB2 copy to be used by your application

Sets the environment required by your application to use a particular DB2 copy or the location specified. If your environment is already set up for the DB2 copy you want to use, you do not need to call this API. If, however, you need to use a different DB2 copy you must call this API. Call this API before loading any DB2 dll files within your process. This call can only be made once per process.

Authorization:

None

Required connection:

None

API include file:

db2ApiInstall.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SelectDB2Copy (
    db2UInt32 versionNumber,
    void *pDB2SelectDB2CopyStruct);
```

```
typedef enum DB2CopyParmType
{
    DB2CopyInvalid=0,
    DB2CopyName,
    DB2CopyPath
} db2CopyParmType;
```

```
typedef struct DB2SelectDB2CopyStruct
{
    DB2CopyParmType Type;
    char *psziDB2Copy;
} db2SelectDB2CopyStruct
```

db2SelectDB2Copy API parameters:

versionNumber

Input. Specifies the version number and release level of the variable passed in as the second parameter, pDB2SelectInstallationStruct.

pDB2SelectDB2CopyStruct

Input. A pointer to the DB2SelectDB2CopyStruct structure.

DB2SelectDB2CopyStruct data structure parameters:

Type Input. This can be either DB2CopyName or DB2CopyPath.

psziDB2Copy

Input. If Type is specified as DB2CopyName, psziDB2Copy is the name of the DB2 copy. If Type is specified as db2CopyPath, psziDB2Copy is the DB2 installation path. This cannot be NULL.

Usage notes:

db2SelectDB2Copy - Select the DB2 copy to be used by your application

To use the API, you will need to include `db2ApiInstall.h`, which will force your application to statically link in `db2ApiInstall.lib`.

In addition, this API must be called before loading any DB2 libraries and can only be called once by an application. You can avoid loading DB2 libraries by making use of the `/delayload` option when linking DB2 libraries or you can load these libraries dynamically using `LoadLibraryEx` and specifying `LOAD_WITH_ALTERED_SEA`.

Related tasks:

- “Selecting a different DB2 copy for your Windows CLI application” in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- “db2swtch - Switch default DB2 copy command” in *Command Reference*

db2SetSyncSession - Set satellite synchronization 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SetSyncSession (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef struct db2SetSyncSessionStruct
{
    char *piSyncSessionID;
} db2SetSyncSessionStruct;
```

db2SetSyncSession 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.

db2SetSyncSessionStruct data structure parameters:

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 526
- "db2GetSyncSession - Get a satellite synchronization session identifier" on page 120
- "db2QuerySatelliteProgress - Get the status of a satellite synchronization session" on page 252
- "db2SyncSatellite - Start satellite synchronization" on page 325

db2SetSyncSession - Set satellite synchronization session

- “db2SyncSatelliteStop - Pause satellite synchronization” on page 326
- “db2SyncSatelliteTest - Test whether a satellite can be synchronized” on page 327

db2SetWriteForDB - Suspend or resume I/O writes for database

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.

Scope:

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

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Database

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SetWriteForDB (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef struct db2SetWriteDbStruct
{
    db2int32 iOption;
    char *piTablespaceNames;
} db2SetWriteDbStruct;
```

db2SetWriteForDB 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 db2SetWriteDbStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2SetWriteDbStruct data structure parameters:

iOption

Input. Specifies the action. Valid values are:

- **DB2_DB_SUSPEND_WRITE**
Suspends I/O write to disk.

db2SetWriteForDB - Suspend or resume I/O writes for database

- DB2_DB_RESUME_WRITE

Resumes I/O write to disk.

piTablespaceNames

Input. Reserved for future use.

Related tasks:

- “Using a split mirror as a backup image” in *Data Recovery and High Availability Guide and Reference*
- “Using a split mirror as a standby database” in *Data Recovery and High Availability Guide and Reference*
- “Using a split mirror to clone a database” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “SET WRITE command” in *Command Reference*
- “SQLCA ” on page 526

db2SpmListIndTrans - List SPM indoubt transactions

Provides a list of transactions that are indoubt at the Syncpoint Manager.

Scope:

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

Authorization:

None

Required connection:

Connection to the Syncpoint Manager

API include file:

sqlxa.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SpmListIndTrans (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2SpmListIndTransStruct
{
    db2SpmRecoverStruct * piIndoubtData;
    db2Uint32             iIndoubtDataLen;
    db2Uint32             oNumIndoubtsReturned;
    db2Uint32             oNumIndoubtsTotal;
    db2Uint32             oReqBufferLen;
} db2XaListIndTransStruct;

typedef SQL_STRUCTURE db2SpmRecoverStruct
{
    SQLXA_XID    xid;
    char         luwid[SQLCSPQY_LUWID_SZ+1];
    char         corrtok[SQLCSPQY_APPLID_SZ+1];
    char         partner[SQLCSPQY_LUNAME_SZ+1];
    char         dbname[SQLCSPQY_DBNAME_SZ+1];
    char         dbalias[SQLCSPQY_DBNAME_SZ+1];
    char         role;
    char         uow_status;
    char         partner_status;
} db2SpmRecoverStruct;
```

db2SpmListIndTrans API parameters:

versionNumber

Input. Specifies the version and release level.

pParmStruct

Input. A pointer to the db2SpmListIndTransStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2SpmListIndTransStruct data structure parameters:

db2SpmListIndTrans - List SPM indoubt transactions

piIndoubtData

Input. A pointer to the application supplied buffer where indoubt data will be returned. The indoubt data is in db2SpmRecoverStruct format. The application can traverse the list of indoubt transactions by using the size of the db2SpmRecoverStruct structure, starting at the address provided by this parameter. If the value is NULL, size of the required buffer is calculated and returned 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 piIndoubtData.

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.

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 piIndoubtData set to NULL. This value can then be used to allocate the required buffer, and the API can be issued with piIndoubtData set to the address of the allocated buffer.

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.

db2SpmRecoverStruct data structure parameters:

xid Output. Specifies the XA identifier assigned by the transaction manager to uniquely identify a global transaction.

luwid Output. Specifies the Logical Unit of Work ID (LUWID) assigned by the Syncpoint Manager to identify the XA Identifier (XID) at the partner system.

corrtok

Output. Specifies the application identifier assigned by the Syncpoint manager for this transaction.

partner

Output. Specifies the name of the Partner system.

dbname

Output. Database of the partner system

dbalias

Output. Specifies the alias of the database where the indoubt transaction is found.

role

Output. Role of the Syncpoint manager.

db2SpmListIndTrans - List SPM indoubt transactions

SQLCSPQY_AR

Syncpoint Manager is an Application Requestor

SQLCSPQY_AS

Syncpoint manager is an Application Server

uow_status

Output. Indicates the status of this indoubt transaction at the Syncpoint Manager. Valid values are:

SQLCSPQY_STATUS_COM

The transaction is in commit status at the Syncpoint Manager. The Transaction is waiting to be resynchronized with the partner system during the next resynchronization interval.

SQLCSPQY_STATUS_RBK

The transaction is in rollback status at the Syncpoint Manager. Waiting for the Partner system to initiate resynchronization and resolve indoubt.

SQLCSPQY_STATUS_IDB

The transaction is in prepared state at the Syncpoint manager. 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.

SQLCSPQY_STATUS_HCM

The transaction has been heuristically committed.

SQLCSPQY_STATUS_HRB

The transaction has been heuristically rolled back.

Usage notes:

A typical application will perform the following steps after setting the current connection to the Syncpoint Manager*:

1. Call db2SpmListIndTrans API 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 of 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.

* To connect to the Syncpoint Manger, determine the name of the Syncpoint Manager being used at the DB2 Connect Server. This can be determined by querying the database configuration parameter, spm_name, at the DB2 Connect Server. Issue a connect by specifying the spm_name as the database alias on the connect API.

Related reference:

- "sqlxphcm - Commit an indoubt transaction" on page 502
- "sqlxphrl - Roll back an indoubt transaction" on page 504
- "SQLCA " on page 526

db2SyncSatellite - Start satellite synchronization

Synchronizes a satellite. Satellite synchronization involves bringing a satellite to a state that is consistent with the other satellites of its group.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SyncSatellite (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2SyncSatellite 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 526
- "db2GetSyncSession - Get a satellite synchronization session identifier" on page 120
- "db2QuerySatelliteProgress - Get the status of a satellite synchronization session" on page 252
- "db2SetSyncSession - Set satellite synchronization session" on page 318
- "db2SyncSatelliteStop - Pause satellite synchronization" on page 326
- "db2SyncSatelliteTest - Test whether a satellite can be synchronized" on page 327

db2SyncSatelliteStop - Pause satellite synchronization

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SyncSatelliteStop (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2SyncSatelliteStop 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:

- "db2SyncSatellite - Start satellite synchronization" on page 325
- "SQLCA " on page 526
- "db2GetSyncSession - Get a satellite synchronization session identifier" on page 120
- "db2QuerySatelliteProgress - Get the status of a satellite synchronization session" on page 252
- "db2SetSyncSession - Set satellite synchronization session" on page 318
- "db2SyncSatelliteTest - Test whether a satellite can be synchronized" on page 327

db2SyncSatelliteTest - Test whether a satellite can be synchronized

Tests the ability of a satellite to synchronize that is, tests whether the satellite can be brought to a state that is consistent with the other satellites of its group.

Authorization:

None

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2SyncSatelliteTest (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);
```

db2SyncSatelliteTest 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 526
- "db2GetSyncSession - Get a satellite synchronization session identifier" on page 120
- "db2QuerySatelliteProgress - Get the status of a satellite synchronization session" on page 252
- "db2SyncSatellite - Start satellite synchronization" on page 325
- "db2SetSyncSession - Set satellite synchronization session" on page 318
- "db2SyncSatelliteStop - Pause satellite synchronization" on page 326

db2UpdateAlertCfg - Update the alert configuration settings for health indicators

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

API and data structure syntax:

```
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
{
```

db2UpdateAlertCfg - Update the alert configuration settings for health indicators

```
    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;
```

db2UpdateAlertCfg 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.

db2UpdateAlertCfgData data structure parameters:

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

db2UpdateAlertCfg - Update the alert configuration settings for health indicators

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.

db2AlertAttrib data structure parameters:

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 are:

- DB2ALERTCFG_ALARM
- DB2ALERTCFG_WARNING
- DB2ALERTCFG_SENSITIVITY
- DB2ALERTCFG_ACTIONS_ENABLED

db2UpdateAlertCfg - Update the alert configuration settings for health indicators

- DB2ALERTCFG_THRESHOLD_CHECK

db2AlertActionUpdate data structure parameters:

iActionType

Input. Specifies the alert action. Valid values are:

- 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-numerical-ID>.<task-numerical-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.

db2AlertActionDelete data structure parameters:

iActionType

Input. Specifies the alert action. Valid values are:

- DB2ALERTCFG_ACTIONTYPE_SCRIPT
- DB2ALERTCFG_ACTIONTYPE_TASK

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>.

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.

db2AlertActionNew data structure parameters:

iActionType

Input. Specifies the alert action. Valid values are:

- DB2ALERTCFG_ACTIONTYPE_SCRIPT

db2UpdateAlertCfg - Update the alert configuration settings for health indicators

- DB2ALERTCFG_ACTIONTYPE_TASK

piScriptAttribs

Input. A pointer to the db2AlertScriptAction structure.

piTaskAttribs

Input. A pointer to the db2AlertTaskAction structure.

db2AlertScriptAction data structure parameters:

scriptType

Specifies the type of 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

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.

db2AlertTaskAction data structure parameters:

pTaskname

The name of the task.

condition

The condition for which to run the action.

db2UpdateAlertCfg - Update the alert configuration settings for health indicators

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.

Related tasks:

- “Configuring health indicators using a client application” in *System Monitor Guide and Reference*

Related reference:

- “UPDATE ALERT CONFIGURATION command” in *Command Reference*
- “UPDATE ALERT CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2GetAlertCfg - Get the alert configuration settings for the health indicators” on page 95
- “db2ResetAlertCfg - Reset the alert configuration of health indicators” on page 279
- “SQLCA ” on page 526

db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory

Updates the alternate server for a database alias in the system database directory.

Scope:

This API affects the system database directory.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
    db2UpdateAlternateServerForDB (
        db2UInt32 versionNumber,
        void * pParmStruct,
        struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2UpdateAltServerStruct
{
    char *piDbAlias;
    char *piHostName;
    char *piPort;
} db2UpdateAltServerStruct;

SQL_API_RC SQL_API_FN
    db2gUpdateAlternateServerForDB (
        db2UInt32 versionNumber,
        void * pParmStruct,
        struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gUpdateAltServerStruct
{
    db2UInt32 iDbAlias_len;
    char *piDbAlias;
    db2UInt32 iHostName_len;
    char *piHostName;
    db2UInt32 iPort_len;
    char *piPort;
} db2gUpdateAltServerStruct;
```

db2UpdateAlternateServerForDB 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 db2UpdateAltServerStruct structure.

db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory

pSqlca

Output. A pointer to the sqlca structure.

db2UpdateAltServerStruct data structure parameters:

piDbAlias

Input. A string containing an alias for the database.

piHostName

Input. A string containing the host name or the IP address of the node where the alternate server for the database resides. The host name is the name of the node that is known to the TCP/IP network. The maximum length of the host name is 255 characters. The IP address can be an IPv4 or an IPv6 address.

piPort Input. The port number of the alternate server database manager instance. The maximum length of the port number is 14 characters.

db2gUpdateAltServerStruct data structure specific parameters:

iDbAlias_len

Input. The length in bytes of piDbAlias.

iHostName_len

Input. The length in bytes of piHostName.

iPort_len

Input. The length in bytes of piPort.

Usage notes:

The API will only be applied to the system database directory.

The API should only be used on a server. If it is issued on a client, it will be ignored and warning SQL1889W will be issued.

If LDAP (Lightweight Directory Access Protocol) support is enabled on the current machine, the alternate server for the database will automatically be updated in the LDAP directory.

Related reference:

- "UPDATE ALTERNATE SERVER FOR DATABASE command" in *Command Reference*
- "db2LdapUpdateAlternateServerForDB - Update the alternate server for the database on the LDAP server" on page 203
- "sqlcadb - Catalog a database in the system database directory" on page 400
- "sqleuncd - Uncatalog a database from the system database directory" on page 478
- "SQLCA " on page 526

db2UpdateContact - Update the attributes of a 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`

API and data structure syntax:

```
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;
```

db2UpdateContact 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.

db2UpdateContactData data structure parameters:

piContactName

Input. Specifies the name of the contact to be updated.

iNumAttribsUpdated

Input. The number attributes to be updated.

db2UpdateContact - Update the attributes of a contact

piAttribs

Input. A pointer to the db2ContactAttrib structure.

db2ContactAttrib data structure parameters:

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.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

Related reference:

- “UPDATE CONTACT command” in *Command Reference*
- “contact_host - Location of contact list configuration parameter” in *Performance Guide*
- “db2AddContact - Add a contact to whom notification messages can be sent” on page 24
- “db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent” on page 77
- “db2GetContacts - Get the list of contacts to whom notification messages can be sent” on page 105
- “SQLCA ” on page 526
- “UPDATE CONTACT command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

db2UpdateContactGroup - Update the attributes of a 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

API and data structure syntax:

```
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;
```

db2UpdateContactGroup 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.

db2UpdateContactGroupData data structure parameters:

piUserid

Input. The user name.

db2UpdateContactGroup - Update the attributes of a contact group

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.

db2ContactTypeData data structure parameters:

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.

Usage notes:

This API is not supported on UNIX and Linux. However, you can access the same functionality through the SQL interface.

Related reference:

- "UPDATE CONTACTGROUP command" in *Command Reference*
- "contact_host - Location of contact list configuration parameter" in *Performance Guide*
- "db2AddContactGroup - Add a contact group to whom notification messages can be sent" on page 26
- "db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent" on page 79
- "db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent" on page 101
- "db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent" on page 103
- "SQLCA " on page 526
- "UPDATE CONTACTGROUP command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent

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

API and data structure syntax:

```
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;
```

db2UpdateHealthNotificationList 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.

db2UpdateHealthNotificationListData data structure parameters:

db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent

iNumUpdates

Input. The number of updates.

piUpdates

Input. A pointer to the db2HealthNotificationListUpdate structure.

db2HealthNotificationListUpdate data structure parameters:

iUpdateType

Input. Specifies the type of update. Valid values are:

- DB2HEALTHNOTIFICATIONLIST_ADD
- DB2HEALTHNOTIFICATIONLIST_DROP

piContact

Input. A pointer to the db2ContactTypeData structure.

db2ContactTypeData data structure parameters:

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 tasks:

- “Enabling health alert notification” in *System Monitor Guide and Reference*

Related reference:

- “UPDATE HEALTH NOTIFICATION CONTACT LIST command” in *Command Reference*
- “db2GetHealthNotificationList - Get the list of contacts to whom health alert notifications can be sent” on page 107
- “SQLCA ” on page 526
- “UPDATE HEALTH NOTIFICATION CONTACT LIST command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

db2UtilityControl - Set the priority level of running utilities

Controls the priority level of running utilities. Can be used to throttle and unthrottle utility invocations.

Authorization:

sysadm

Required connection:

Instance

API include file:

db2ApiDf.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2UtilityControl (
    db2UInt32 version,
    void * pUtilityControlStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2UtilityControlStruct
{
    db2UInt32 iID;
    db2UInt32 iAttribute;
    void *pioValue;
} db2UtilityControlStruct;

SQL_API_RC SQL_API_FN
db2gUtilityControl (
    db2UInt32 version,
    void * pgUtilityControlStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gUtilityControlStruct
{
    db2UInt32 iID;
    db2UInt32 iAttribute;
    void *pioValue;
} db2gUtilityControlStruct;
```

db2UtilityControl API parameters:

version

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

pUtilityControlStruct

Input. A pointer to the db2UtilityControlStruct structure.

pSqlca

Output. A pointer to the sqlca structure.

db2UtilityControlStruct data structure parameters:

iID Input. Specifies the ID of the utility to modify.

db2UtilityControl - Set the priority level of running utilities

iAttribute

Input. Specifies the attribute to modify. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2UTILCTRL_PRIORITY_ATTRIB

Modify the throttling priority of the utility.

pioValue

Input. Specifies the new attribute value associated with the iAttribute parameter.

Note: If the iAttribute parameter is set to DB2UTILCTRL_PRIORITY_ATTRIB, then the pioValue parameter must point to a db2UInt32 containing the priority.

Usage notes:

SQL1153N will be returned if there is no existing utility with the specified iId. This may indicate that the function was invoked with invalid arguments or that the utility has completed.

SQL1154N will be returned if the utility does not support throttling.

Related reference:

- "SQLCA " on page 526

db2XaGetInfo - Get information for a resource manager

Extracts information for a particular resource manager once an xa_open call has been made.

Authorization:

Instance - SPM name connection

Required Connection:

Database

API include file:

sqlxa.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
db2XaGetInfo(db2UInt32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);
```

```
typedef SQL_STRUCTURE db2XaGetInfoStruct
{
    db2int32 iRmid;
    struct sqlca oLastSqlca;
} db2XaGetInfoStruct;
```

db2XaGetInfo 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.

db2XaGetInfoStruct data structure parameters:

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 526

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:

None

Required connection:

Database

API include file:

sqlxa.h

API and data structure syntax:

```
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[SQLXA_USERID_SZ];
    char log_full;
    char connected;
    char indoubt_status;
    char originator;
    char reserved[8];
    sqluint32 rmn;
    rm_entry rm_list[SQLXA_MAX_FedRM];
} db2XaRecoverStruct;

typedef SQL_STRUCTURE rm_entry
{
    char name[SQLQG_MAX_SERVER_NAME_LEN];
    SQLXA_XID xid;
} rm_entry;
```

db2XaListIndTrans API parameters:

db2XaListIndTrans - List indoubt transactions

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.

db2XaListIndTransStruct data structure parameters:

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.

iIndoubtDataLen

Input. Size of the buffer pointed to by piIndoubtData parameter in bytes.

oNumIndoubtsReturned

Output. The number of indoubt transaction records returned in the buffer specified by piIndoubtData.

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 piIndoubtData set to NULL. This value can then be used to allocate the required buffer, and the API can be issued with piIndoubtData 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.

db2XaRecoverStruct data structure parameters:

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.

connected Indicates whether an application is connected.

Possible values for CONNECTED (defined in sqlxa) are:

SQLXA_TRUE

True. The transaction is undergoing normal syncpoint processing, and is waiting for the second phase of the two-phase commit.

SQLXA_FALSE

False. The transaction was left indoubt by an earlier failure, and is now waiting for re-sync from a transaction manager.

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

db2XaListIndTrans - List indoubt transactions

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.

When the originator parameter is set to the value `SQLXA_ORIG_FXA`, valid values for the `indoubt_status` parameter (defined in `sqlxa.h` located in the include directory) are:

SQLXA_TS_MFCACK

Indicates that the transaction is missing commit acknowledgement from one or more federated data sources.

SQLXA_TS_MFRACK

Indicates that the transaction is missing rollback acknowledgement from one or more federated data sources.

originator

Identifies the origin of an indoubt transaction.

Possible values for `ORIGINATOR` (defined in `sqlxa.h` located in the include directory) are:

SQLXA_ORIG_PE

Transaction originated by DB2 in MPP environment.

SQLXA_ORIG_XA

Transaction originated by XA.

SQLXA_ORIG_FXA

Transaction originated in the second phase of the federated two-phase commit process. It indicates that this transaction has entered the second phase of the two-phase commit protocol, however one or more federated data sources cannot complete the second phase or cannot communicate with the federated server.

reserved

The first byte is used to indicate the type of indoubt transaction: 0 indicates RM, and 1 indicates TM.

rmn Output. Number of federated data sources that failed to commit or rollback a transaction.

rm_list

Output. List of failed federated data source entries, each of which contains a server name and a `xid`.

rm_entry data structure parameters:

name Output. Name of a federated data source.

xid Output. Specifies the XA identifier assigned by the federated database to a federated data source to uniquely identify a federated transaction.

Usage notes:

`SQLXA_MAX_FEDRM` is defined to be 16. Most federated transactions involve less than 10 data sources. If more than 16 federated data sources fail to commit or rollback in a transaction, only 16 of them will be returned by the `db2XaListIndTrans` API for this indoubt transaction. For a non-federated indoubt transaction, `rmn` parameter will be set to 0, indicating that the indoubt transaction involves no federated data sources.

db2XaListIndTrans - List indoubt transactions

If a federated indoubt transaction involves more than 16 failed federated data sources, when the heuristic processing is invoked, all the data sources (regardless of whether they are returned by the db2XaListIndTrans API) will commit or roll back the indoubt transaction. Any federated data source that successfully committed or rolled back the indoubt transaction will be removed from the list of failed federated data sources for the federated indoubt transaction. On the next call to the db2XaListIndTrans API, only federated data sources that still failed to commit or roll back the indoubt transaction will remain in the list for the federated indoubt transaction.

To obtain the list of data sources in a federated indoubt transaction, you must compile applications using DB2 Version 9.1 header files and pass in a version number db2Version900 or higher (for later releases) to the db2XaListIndTrans API. If you pass in a lower version number, the API will still return a list of indoubt transactions, but federated data source information will be excluded. Regardless, the version of the header file used by the application must be in sync with the version number passed to the API. Otherwise, the results will be unpredictable.

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 concepts:

- “Indoubt transaction management APIs” in *Administration Guide: Planning*

Related tasks:

- “Resolving indoubt transactions manually” in *Administration Guide: Planning*

Related reference:

- “LIST INDOUBT TRANSACTIONS command” in *Command Reference*
- “sqlxphcm - Commit an indoubt transaction” on page 502
- “sqlxphrl - Roll back an indoubt transaction” on page 504
- “SQLCA ” on page 526
- “sqlcspqy - List DRDA indoubt transactions” on page 381
- “sqlxhfrg - Forget transaction status” on page 501

sqlabndx - Bind application program to create a package

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlabndx (
    _SQLOLDCHAR * pBindFileName,
    _SQLOLDCHAR * pMsgFileName,
    struct sqlopt * pBindOptions,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgbndx (
    unsigned short MsgFileNameLen,
    unsigned short BindFileNameLen,
    struct sqlca * pSqlca,
    struct sqlopt * pBindOptions,
    _SQLOLDCHAR * pMsgFileName,
    _SQLOLDCHAR * pBindFileName);
```

sqlabndx API parameters:

pBindFileName

db2XaListIndTrans - List indoubt transactions

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.

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.

pBindOptions

Input. A structure used to pass bind options to the API. For more information about this structure, see SQLOPT.

pSqlca

Output. A pointer to the sqlca structure.

sqlgbndx API-specific parameters:

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.

BindFileNameLen

Input. Length in bytes of the pBindFileName parameter.

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.)

db2XaListIndTrans - List indoubt transactions

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 Version 8 client to a Version 8 server, and then executed against a Version 7 server.

The Bind option types and values are defined in sql.h.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related reference:

- “sqlarbnd - Rebind package” on page 358
- “sqlaprep - Precompile application program” on page 355
- “SQLCA ” on page 526
- “sqlchar ” on page 527
- “sqlopt ” on page 579
- “BIND command” in *Command Reference*

Related samples:

- “dbsample.sqc -- Creates a sample database (C)”
- “dbpkg.sqc -- How to work with packages (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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlaintp (
    char * pBuffer,
    short BufferSize,
    short LineWidth,
    const char * pMsgFileName,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgintp (
    short BufferSize,
    short LineWidth,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pBuffer);
```

sqlaintp API parameters:

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.

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.

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.

sqlaintp - Get error 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.

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.

Related reference:

- "sqlogstt - Get the SQLSTATE message" on page 485
- "SQLCA " on page 526

Related samples:

- "dbcfg.sqlC -- Configure database and database manager configuration parameters (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"
- "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)"
- "dbcfg.sqlc -- Configure database and database manager configuration parameters (C)"

sqlaprep - Precompile application 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlaprep (
    _SQLOLDCHAR * pProgramName,
    _SQLOLDCHAR * pMsgFileName,
    struct sqlopt * pPrepOptions,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgprep (
    unsigned short MsgFileNameLen,
    unsigned short ProgramNameLen,
    struct sqlca * pSqlca,
    struct sqlopt * pPrepOptions,
    _SQLOLDCHAR * pMsgFileName,
    _SQLOLDCHAR * pProgramName);
```

sqlaprep API parameters:

sqlaprep - Precompile application program

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.

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.

pPrepOptions

Input. A structure used to pass precompile options to the API. For more information about this structure, see SQLOPT.

pSqlca

Output. A pointer to the sqlca structure.

sqlgprep API-specific parameters:

MsgFileNameLen

Input. Length in bytes of the pMsgFileName parameter.

ProgramNameLen

Input. Length in bytes of the pProgramName parameter.

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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related reference:

- “sqlabndx - Bind application program to create a package” on page 350
- “SQLCA ” on page 526
- “sqlopt ” on page 579
- “PRECOMPILE command” in *Command Reference*

Related samples:

- “dbpkg.sqc -- How to work with packages (C)”
- “dbpkg.sqC -- How to work with packages (C++)”

sqlarbind - Rebind package

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlarbind (
    char * pPackageName,
    struct sqlca * pSqlca,
    struct sqlopt * pRebindOptions);
```

```
SQL_API_RC SQL_API_FN
sqlgrbind (
    unsigned short PackageNameLen,
    char * pPackageName,
    struct sqlca * pSqlca,
    struct sqlopt * pRebindOptions);
```

sqlarbind API parameters:

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_VERSION_OPT 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.

sqlgrbind API-specific parameters:

PackageNameLen

Input. Length in bytes of the pPackageName parameter.

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.fs. 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.

sqlarbnd - Rebind package

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_VERSION_OPT 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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- "Calling DB2 APIs in REXX" on page 754

Related tasks:

- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:

- "db2rbind - Rebind all packages command" in *Command Reference*
- "REBIND command" in *Command Reference*
- "REBIND_ROUTINE_PACKAGE procedure" in *Administrative SQL Routines and Views*
- "db2Runstats - Update statistics for tables and indexes" on page 306
- "sqlabndx - Bind application program to create a package" on page 350
- "sqlaprep - Precompile application program" on page 355
- "sqlopt " on page 579
- "SQLCA " on page 526

Related samples:

- "dbpkg.sqc -- How to work with packages (C)"
- "dbpkg.sqC -- How to work with packages (C++)"
- "rebind.sqb -- How to rebind a package (IBM COBOL)"

sqlbctcq - Close a 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbctcq (
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgctcq (
    struct sqlca * pSqlca);
```

sqlbctcq API parameters:

pSqlca

Output. A pointer to the sqlca structure.

Related reference:

- “sqlbftcq - Fetch the query data for rows in a table space container” on page 363
- “sqlbotcq - Open a table space container query” on page 371
- “sqlbstsc - Set table space containers” on page 377
- “sqlbtcq - Get the query data for all table space containers” on page 379
- “SQLCA ” on page 526

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)”
- “tinfo.sqc -- How to get information at the table space level (C)”
- “tinfo.sqC -- How to get information at the table space level (C++)”

sqlbctsq - Close a 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN  
sqlbctsq (  
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN  
sqlgctsq (  
    struct sqlca * pSqlca);
```

sqlbctsq API parameters:

pSqlca

Output. A pointer to the sqlca structure.

Related reference:

- “sqlbftpq - Fetch the query data for rows in a table space” on page 365
- “sqlbgtss - Get table space usage statistics” on page 367
- “sqlbmtsq - Get the query data for all table spaces” on page 369
- “sqlbotsq - Open a table space query” on page 373
- “sqlbstpq - Get information about a single table space” on page 375
- “SQLCA ” on page 526

Related samples:

- “tabspace.sqb -- How to get tablespace information (IBM COBOL)”
- “tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)”
- “tinfo.sqc -- How to get information at the table space level (C)”
- “tinfo.sqC -- How to get information at the table space level (C++)”

sqlbftcq - Fetch the query data for rows in a table space container

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbftcq (
    struct sqlca * pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData,
    sqluint32 * pNumContainers);
```

```
SQL_API_RC SQL_API_FN
sqlgftcq (
    struct sqlca * pSqlca,
    sqluint32 MaxContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData,
    sqluint32 * pNumContainers);
```

sqlbftcq 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.

sqlbftcq - Fetch the query data for rows in a table space container

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:

- “`sqlbctcq` - Close a table space container query” on page 361
- “`sqlbotcq` - Open a table space container query” on page 371
- “`sqlbstsc` - Set table space containers” on page 377
- “`sqlbtcq` - Get the query data for all table space containers” on page 379
- “`SQLB_TBSCONTQRY_DATA` ” on page 519
- “`SQLCA` ” on page 526

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 the query data for rows in a table space

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbftpq (
    struct sqlca * pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 * pNumTablespaces);
```

```
SQL_API_RC SQL_API_FN
sqlgftpq (
    struct sqlca * pSqlca,
    sqluint32 MaxTablespaces,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 * pNumTablespaces);
```

sqlbftpq 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

sqlbftpq - Fetch the query data for rows in a table space

- 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:

- “sqlbctsq - Close a table space query” on page 362
- “sqlbgtss - Get table space usage statistics” on page 367
- “sqlbmtsq - Get the query data for all table spaces” on page 369
- “sqlbotsq - Open a table space query” on page 373
- “sqlbstpq - Get information about a single table space” on page 375
- “SQLB_TBSPQRY_DATA ” on page 521
- “SQLCA ” on page 526

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 usage 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:

Database

API include file:

sqlutil.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbgtss (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS * pTablespaceStats);
```

```
SQL_API_RC SQL_API_FN
sqlggtss (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBS_STATS * pTablespaceStats);
```

sqlbgtss 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:

- “sqlbctsq - Close a table space query” on page 362

sqlbgtss - Get table space usage statistics

- “sqlbftpq - Fetch the query data for rows in a table space” on page 365
- “sqlbmtsq - Get the query data for all table spaces” on page 369
- “sqlbotsq - Open a table space query” on page 373
- “sqlbstpq - Get information about a single table space” on page 375
- “SQLB_TBS_STATS ” on page 517
- “SQLCA ” on page 526

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 - Get the query data for all table spaces

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbmtsq (
    struct sqlca * pSqlca,
    sqluint32 * pNumTablespaces,
    struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
```

```
SQL_API_RC SQL_API_FN
sqlgmtsq (
    struct sqlca * pSqlca,
    sqluint32 * pNumTablespaces,
    struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
    sqluint32 reserved1,
    sqluint32 reserved2);
```

sqlbmtsq 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.

sqlbmtsq - Get the query data for all table spaces

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 sqlfmem.

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:

- "LIST TABLESPACES command" in *Command Reference*
- "sqlbctsq - Close a table space query" on page 362
- "sqlbftpq - Fetch the query data for rows in a table space" on page 365
- "sqlbgtss - Get table space usage statistics" on page 367
- "sqlbotsq - Open a table space query" on page 373
- "sqlbstpq - Get information about a single table space" on page 375
- "sqlfmem - Free the memory allocated by the sqlbctq and sqlbmtsq API" on page 432
- "SQLCA " on page 526
- "sqlbctq - Get the query data for all table space containers" on page 379

Related samples:

- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "tinfo.sqC -- How to get information at the table space level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

sqlbotcq - Open a 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

Required connection:

Database

API include file:

sqlutil.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbotcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers);
```

```
SQL_API_RC SQL_API_FN
sqlgotcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers);
```

sqlbotcq 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:

- sqlbctcq
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

sqlbotcq - Open a table space container query

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 sqlfmem 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 sqlbctcq).

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 (sqlbctcq 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:

- “sqlbctcq - Close a table space container query” on page 361
- “sqlbftcq - Fetch the query data for rows in a table space container” on page 363
- “sqlbstsc - Set table space containers” on page 377
- “SQLCA ” on page 526
- “sqlbctcq - Get the query data for all table space containers” on page 379

Related samples:

- “tabscont.sqb -- How to get tablespace container information (IBM COBOL)”
- “tspc.sqb -- How to copy and free memory in a tablespace (IBM COBOL)”
- “tinfo.sqc -- How to get information at the table space level (C)”
- “tinfo.sqC -- How to get information at the table space level (C++)”

sqlbotsq - Open a 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbotsq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 * pNumTablespaces);
```

```
SQL_API_RC SQL_API_FN
sqlgotsq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceQueryOptions,
    sqluint32 * pNumTablespaces);
```

sqlbotsq 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.

sqlbotsq - Open a table space query

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 `sqlfmem` 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:

- “`sqlbctsq` - Close a table space query” on page 362
- “`sqlbftpq` - Fetch the query data for rows in a table space” on page 365
- “`sqlbmtsq` - Get the query data for all table spaces” on page 369
- “`sqlbstpq` - Get information about a single table space” on page 375
- “SQLCA ” on page 526

Related samples:

- “`tabspace.sqb` -- How to get tablespace information (IBM COBOL)”
- “`tspc.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 - Get information about a single table space

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbstpq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 reserved);
```

```
SQL_API_RC SQL_API_FN
sqlgstpq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    sqluint32 reserved);
```

sqlbstpq 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:

sqlbstpq - Get information about a single table space

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.

Related reference:

- “sqlbctsq - Close a table space query” on page 362
- “sqlbftpq - Fetch the query data for rows in a table space” on page 365
- “sqlbgtss - Get table space usage statistics” on page 367
- “sqlbmtsq - Get the query data for all table spaces” on page 369
- “sqlbotsq - Open a table space query” on page 373
- “SQLCA ” on page 526

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbstsc (
    struct sqlca * pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData);
```

```
SQL_API_RC SQL_API_FN
sqlgstsc (
    struct sqlca * pSqlca,
    sqluint32 SetContainerOptions,
    sqluint32 TablespaceId,
    sqluint32 NumContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData);
```

sqlbstsc 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.

sqlbstsc - Set table space containers

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 sqlurstm 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:

- "SET TABLESPACE CONTAINERS command" in *Command Reference*
- "db2Backup - Back up a database or table space" on page 41
- "db2Restore - Restore a database or table space" on page 284
- "db2Rollforward - Roll forward a database" on page 296
- "SQLCA " on page 526

Related samples:

- "tabscont.sqlb -- How to get tablespace container information (IBM COBOL)"
- "tspc.sqlb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

sqlbtcq - Get the query data for all table space containers

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

Required connection:

Database

API include file:

sqlutil.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlbtcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers,
    struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
```

```
SQL_API_RC SQL_API_FN
sqlgtcq (
    struct sqlca * pSqlca,
    sqluint32 TablespaceId,
    sqluint32 * pNumContainers,
    struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
```

sqlbtcq 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

sqlbtcq - Get the query data for all table space containers

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:

- "LIST TABLESPACE CONTAINERS command" in *Command Reference*
- "sqlbctcq - Close a table space container query" on page 361
- "sqlbftcq - Fetch the query data for rows in a table space container" on page 363
- "sqlbotcq - Open a table space container query" on page 371
- "sqlbstsc - Set table space containers" on page 377
- "sqlefmem - Free the memory allocated by the sqlbtcq and sqlbmtsq API" on page 432
- "SQLCA " on page 526
- "sqlbmtsq - Get the query data for all table spaces" on page 369

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)"
- "tinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "tinfo.sqC -- How to get information at the table space level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

sqlcspqy - List DRDA indoubt transactions

Provides a list of transactions that are indoubt between the syncpoint manager partner connections. This API is being deprecated. Please see 'db2SpmListIndTrans API - List SPM Indoubt Transactions'.

Authorization:

None

Required connection:

Instance

API include file:

sqlxa.h

API and data structure syntax:

```
extern int SQL_API_FN sqlcspqy(SQLCSPQY_INDOUBT    **indoubt_data,
                               sqlint32 *indoubt_count,
                               struct sqlca *sqlca);
```

sqlcspqy 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 the LIST DRDA INDOUBT TRANSACTIONS command, the application process must be connected to the Sync Point Manager (SPM) instance. Use the spm_name database manager configuration parameter as the dbalias on the CONNECT statement.

Related concepts:

- "Indoubt transaction management APIs" in *Administration Guide: Planning*

Related tasks:

sqlcspqy - List DRDA indoubt transactions

- “Resolving indoubt transactions manually” in *Administration Guide: Planning*

Related reference:

- “LIST DRDA INDOUBT TRANSACTIONS command” in *Command Reference*
- “CONNECT (Type 1) statement” in *SQL Reference, Volume 2*
- “CONNECT (Type 2) statement” in *SQL Reference, Volume 2*
- “spm_name - Sync point manager name configuration parameter” in *Performance Guide*
- “db2SpmListIndTrans - List SPM indoubt transactions” on page 322
- “db2XaListIndTrans - List indoubt transactions” on page 345
- “SQLCA ” on page 526

sqlc_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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlc_activate_db (
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
```

```
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);
```

sqlc_activate_db API parameters:

pDbAlias

Input. Pointer to the database alias name.

pUserName

Input. Pointer to the user ID starting the database. Can be NULL.

sqlc_activate_db - Activate database

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.

sqlc_activate_db API-specific 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.

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 sqlc_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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754
- “Quick-start tips for performance tuning” in *Performance Guide*

Related tasks:

sqlc_activate_db - Activate database

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*

Related reference:

- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “sqlc_deactivate_db - Deactivate database” on page 386
- “SQLCA ” on page 526
- “ACTIVATE DATABASE command” in *Command Reference*

sqlc_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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlc_deactivate_db (
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
```

```
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);
```

sqlc_deactivate_db API parameters:

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.

sqlc_deactivate_db API-specific 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.

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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754
- “Quick-start tips for performance tuning” in *Performance Guide*

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*

Related reference:

- “sqlc_activate_db - Activate database” on page 383
- “SQLCA ” on page 526
- “DEACTIVATE DATABASE command” in *Command Reference*

sqlleadn - Add a database partition server to the partitioned database environment

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlleadn (
    void * pAddNodeOptions,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgaddn (
    unsigned short addnOptionsLen,
    struct sqlca * pSqlca,
    void * pAddNodeOptions);
```

sqlleadn API parameters:

pAddNodeOptions

Input. A pointer to the optional `sqlc_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.

pSqlca

Output. A pointer to the `sqlca` structure.

sqlgaddn API-specific parameters:

sqleaddn - Add a database partition server to the partitioned database environment

addnOptionsLen

Input. A 2-byte unsigned integer representing the length of the optional `sqle_addn_options` structure in bytes.

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.

This API will fail if at any time in a database in the system a user table with an XML column had been, successfully or not, created or an XSR object had been, successfully or not, registered.

To determine whether or not a database is enabled for automatic storage, the `sqleaddn` API has to communicate with the catalog partition for each of the databases in the instance. If automatic storage is enabled then the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the `sqleaddn` API 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 automatic storage and table space definitions. If this time is exceeded, the API fails. Increase the value of `start_stop_time`, and call the API again.

REXX API syntax:

This API can be called from REXX through the `SQLDB2` interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*

sqleaddn - Add a database partition server to the partitioned database environment

Related reference:

- “sqlecrea - Create database” on page 408
- “sqledrpn - Check whether a database partition server can be dropped” on page 427
- “SQLCA ” on page 526
- “sqle_addn_options ” on page 533
- “ALTER DATABASE PARTITION GROUP statement” in *SQL Reference, Volume 2*

sqleatcp - Attach to instance 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqleatcp (
    char * pNodeName,
    char * pUserName,
    char * pPassword,
    char * pNewPassword,
    struct sqlca * pSqlca);
```

sqleatcp API parameters:

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.

pUserName

Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

pPassword

Input. A string containing the password for the specified user name. May be NULL.

pNewPassword

Input. A string containing the new password for the specified user name. Set to NULL if a password change is not required.

pSqlca

Output. A pointer to the sqlca structure.

sqlcatcp - Attach to instance and change password

Usage notes:

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 database 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 sqlsetc API.

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.

Related reference:

- "ATTACH command" in *Command Reference*
- "sqlcatin - Attach to instance" on page 394
- "sqledtin - Detach from instance" on page 429
- "sqlsetc - Set client connection settings" on page 470
- "SQLCA " on page 526

Related samples:

sqleatcp - Attach to instance and change password

- “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 to instance

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqleatin (
    char * pNodeName,
    char * pUserName,
    char * pPassword,
    struct sqlca * pSqlca);
```

```
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);
```

sqleatin API parameters:

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. Can be NULL.

pUserName

Input. A string containing the user name under which the attachment is to be authenticated. Can be NULL.

pPassword

Input. A string containing the password for the specified user name. Can be NULL.

pSqlca

Output. A pointer to the sqlca structure.

sqlgatin API-specific 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.

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 database 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 sqlsetc API.

sqleatin - Attach to instance

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.

Related tasks:

- “Attaching to and detaching from a non-default instance of the database manager” in *Administration Guide: Implementation*

Related reference:

- “ATTACH command” in *Command Reference*
- “SQLCA ” on page 526
- “sqleatcp - Attach to instance and change password” on page 391
- “sqledtin - Detach from instance” on page 429
- “sqlesetc - Set client connection settings” on page 470

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++)”

sqlAttachToCtx - 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlAttachToCtx (
    void * pCtx,
    void * reserved,
    struct sqlca * pSqlca);
```

sqlAttachToCtx API parameters:

pCtx Input. A valid context previously allocated by sqlBeginCtx.

reserved

Reserved for future use. Must be set to NULL.

pSqlca

Output. A pointer to the sqlca structure.

Related reference:

- “sqlBeginCtx - Create and attach to an application context” on page 398
- “SQLCA ” on page 526
- “sqlDetachFromCtx - Detach from context” on page 422
- “sqlEndCtx - Detach from and free the memory associated with an application context” on page 430
- “sqlGetCurrentCtx - Get current context” on page 447
- “sqlInterruptCtx - Interrupt context” on page 450
- “sqlSetTypeCtx - Set application context type” on page 476

Related samples:

- “dbthrsd.sqlc -- How to use multiple context APIs on UNIX (C)”
- “dbthrsd.sqlc -- How to use multiple context APIs on UNIX (C++)”

sqlcBeginCtx - 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 the `sqlcAttachToCtx` API). 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`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlcBeginCtx (
    void ** ppCtx,
    sqlint32 lOptions,
    void * reserved,
    struct sqlca * pSqlca);
```

sqlcBeginCtx 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 `sqlcAttachToCtx` 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.

pSqlca

Output. A pointer to the `sqlca` structure.

Related reference:

- “`sqlcAttachToCtx` - Attach to context” on page 397
- “SQLCA ” on page 526

sqlBeginCtx - Create and attach to an application context

- “sqlDetachFromCtx - Detach from context” on page 422
- “sqlEndCtx - Detach from and free the memory associated with an application context” on page 430
- “sqlGetCurrentCtx - Get current context” on page 447
- “sqlInterruptCtx - Interrupt context” on page 450
- “sqlSetTypeCtx - Set application context type” on page 476

Related samples:

- “dbthrs.sqC -- How to use multiple context APIs on UNIX (C)”
- “dbthrs.sqC++ -- How to use multiple context APIs on UNIX (C++)”

sqlcadb - Catalog a database in the system database directory

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:

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlcadb (
    _SQLOLDCHAR * pDbName,
    _SQLOLDCHAR * pDbAlias,
    unsigned char Type,
    _SQLOLDCHAR * pNodeName,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    unsigned short Authentication,
    _SQLOLDCHAR * pPrincipal,
    struct sqlca * pSqlca);
```

```
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);
```

sqlcadb API parameters:

pDbName

Input. A string containing the database name.

sqlcadb - Catalog a database in the system database directory

pDbAlias

Input. A string containing an alias for the database.

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.

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*.

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*.

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.

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_KERBEROS

Specifies that authentication takes place using Kerberos Security Mechanism.

SQL_AUTHENTICATION_NOT_SPECIFIED

Authentication not specified.

sqlcadb - Catalog a database in the system database directory

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.

SQL_AUTHENTICATION_DATAENC

Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption.

SQL_AUTHENTICATION_GSSPLUGIN

Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

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.

pPrincipal

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_KERBEROS`.

pSqlca

Output. A pointer to the `sqlca` structure.

sqlcadb API-specific 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_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.

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_KERBEROS`.

Usage notes:

sqlcadb - Catalog a database in the system database directory

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.

REXX API syntax:

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]
CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [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.

sqlcadb - Catalog a database in the system database directory

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.

KERBEROS

Specifies that authentication takes place using Kerberos Security Mechanism.

NOT_SPECIFIED

Authentication not specified.

SVR_ENCRYPT

Specifies that authentication takes place on the node containing the target database, and that the authentication password is to be encrypted.

DATAENC

Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption.

GSSPLUGIN

Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

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.

db_global_name

The fully qualified name that uniquely identifies the database in the DCE name space.

DCE The global directory service being used.

REXX examples:

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"
```

Related tasks:

- “Cataloging a database” in *Administration Guide: Implementation*

Related reference:

- “CATALOG DATABASE command” in *Command Reference*

sqlcadb - Catalog a database in the system database directory

- “db2DbDirCloseScan - End a system or local database directory scan” on page 69
- “db2DbDirOpenScan - Start a system or local database directory scan” on page 75
- “sqlcrea - Create database” on page 408
- “sqluncd - Uncatalog a database from the system database directory” on page 478
- “SQLCA ” on page 526
- “db2DbDirGetNextEntry - Get the next system or local database directory entry” on page 71

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++)”

sqlcran - Create a database on a database partition server

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlcran (
    char * pDbName,
    void * pReserved,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgcran (
    unsigned short reservedLen,
    unsigned short dbNameLen,
    struct sqlca * pSqlca,
    void * pReserved,
    char * pDbName);
```

sqlcran API parameters:

pDbName

Input. A string containing the name of the database to be created. Must not be NULL.

pReserved

Input. A spare pointer that is set to null or points to zero. Reserved for future use.

sqlcran - Create a database on a database partition server

pSqlca

Output. A pointer to the sqlca structure.

sqlgcran API-specific 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.

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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*

Related reference:

- “CREATE DATABASE command” in *Command Reference*
- “db2Restore - Restore a database or table space” on page 284
- “sqlcrea - Create database” on page 408
- “sqledpan - Drop a database on a database partition server” on page 423
- “SQLCA ” on page 526

sqlcrea - 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. To use XML features, you must set the SQLDBCODESET field of the sqlldbterritoryinfo data structure to UTF-8.

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlcrea (
    char * pDbName,
    char * pLocalDbAlias,
    char * pPath,
    struct sqldbdesc * pDbDescriptor,
    SQLEDBTERRITORYINFO * pTerritoryInfo,
    char Reserved2,
    void * pDbDescriptorExt,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgcrea (
    unsigned short PathLen,
    unsigned short LocalDbAliasLen,
    unsigned short DbNameLen,
    struct sqlca * pSqlca,
    void * pReserved1,
    unsigned short Reserved2,
    SQLEDBTERRITORYINFO * pTerritoryInfo,
    struct sqldbdesc * pDbDescriptor,
    char * pPath,
    char * pLocalDbAlias,
    char * pDbName);
```

sqlcrea API parameters:

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.

pLocalDbAlias

Input. A string containing the alias to be placed in the client's system database directory. Can be NULL. If no local alias is specified, the database name is the default.

pPath Input. On Linux and UNIX 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. Can 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.

pDbDescriptor

Input. A pointer to the database description block that is used when creating the database. The database description block can be used by you to supply values that are permanently stored in the configuration file of the database.

The supplied values are a collating sequence, a database comment, or a table space definition. The supplied value can be NULL if you do not want to supply any values. For information about the values that can be supplied through this parameter, see the SQLEDBDESC data structure topic.

pTerritoryInfo

Input. A pointer to the sqledbterritoryinfo structure, containing the locale and the code set for the database. Can be NULL. In a future release of the DB2 database system, the default code set will be changed to UTF-8 when creating a database. If a particular code set and territory is needed for a database, the desired code set and territory should be specified via the sqledbterritoryinfo structure.

Reserved2

Input. Reserved for future use.

pDbDescriptorExt

Input. This parameter refers to an extended database description block (sqledbdesext) that is used when creating the database. The extended database description block controls automatic storage for a database, chooses a default page size for the database, and specifies values for new table space attributes that have been introduced. If set to null or zero, a default page size of 4 096 bytes is chosen for the database and automatic storage is enabled.

pSqlca

Output. A pointer to the sqlca structure.

sqlcrea - Create database

sqlgcrea API-specific 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.

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 `$DB2INSTANCE/NODExxxx` directory under the specified subdirectory at each database partition server, where `xxxx` represents the local database partition server number. In a single-partition environment, creates a `$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, `sqlcrea` 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/NODExxxx, under the specified or default path on all database partition servers. The xxx is the node number as defined in the db2nodes.cfg file (that is, node 0 becomes NODE0000). Subdirectories SQL00001 through SQLnnnnn 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).

On Windows and AIX operating systems, the length of the code set name is limited to a maximum of 9 characters. For example, specify a code set name such as ISO885915 instead of ISO8859-15.

Execution of the CREATE DATABASE command 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 most prominent value 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).

sqlcrea - Create database

sql850b

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).

sql932a

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).

sql932b

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 sqlcadb to define different alias names for the new database.

The Configuration Advisor is called by default during the database creation process unless specifically told not to do so.

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

For non-Unicode databases, this is the default option, with the collating sequence based on the database territory. For Unicode databases, this option is equivalent to the IDENTITY option.

IDENTITY

Identity collating sequence, in which strings are compared byte for byte. This is the default for Unicode databases.

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 directories (tablespace containers) that will be created and used to store the database table files for any default SMS table spaces.

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)

sqlcrea - Create database

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 4 KB 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.

Related tasks:

- “Creating a database” in *Administration Guide: Implementation*

Related reference:

- “CREATE DATABASE command” in *Command Reference*
- “sqlbdbesext ” on page 556
- “sqlcadb - Catalog a database in the system database directory” on page 400
- “sqlcran - Create a database on a database partition server” on page 406
- “sqldbterritoryinfo ” on page 563
- “sqldrpd - Drop database” on page 425
- “SQLCA ” on page 526
- “sqlbdbesc ” on page 549
- “sqldpan - Drop a database on a database partition server” on page 423

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)”
- “dbsample.sqc -- Creates a sample database (C)”
- “dbrecov.sqc -- How to recover a database (C)”
- “dbcreate.C -- Create and drop databases (C++)”
- “dbrecov.sqC -- How to recover a database (C++)”

sqlctnd - Catalog an entry in the node directory

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlctnd (
    struct sqlc_node_struct * pNodeInfo,
    void * pProtocolInfo,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgctnd (
    struct sqlca * pSqlca,
    struct sqlc_node_struct * pNodeInfo,
    void * pProtocolInfo);
```

sqlctnd API parameters:

pNodeInfo

Input. A pointer to a node directory structure.

pProtocolInfo

Input. A pointer to the protocol structure:

- SQLE-NODE-LOCAL
- SQLE-NODE-NPIPE
- SQLE-NODE-TCPIP

pSqlca

Output. A pointer to the sqlca structure.

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

sqlectnd - Catalog an entry in the node directory

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 command) and then restart (db2start command) the database manager. To refresh the directory cache for another application, stop and then restart that application.

REXX API syntax, option 1:

```
CATALOG LOCAL NODE nodename INSTANCE instance_name [WITH comment]
```

REXX API parameters, option 1:

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, option 2:

```
CATALOG NPIPE NODE nodename REMOTE computer_name INSTANCE instance_name
```

REXX API parameters, option 2:

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, option 3:

```
CATALOG TCP/IP NODE nodename REMOTE hostname SERVER servicename  
[WITH comment]
```

REXX API parameters, option 3:

nodename

Alias for the node to be cataloged.

hostname

Host name or IPv4 address or IPv6 address 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.

REXX API syntax, option 4:

```
CATALOG TCP/IP4 NODE nodename REMOTE hostname SERVER servicename  
[WITH comment]
```

sqlcnd - Catalog an entry in the node directory

REXX API parameters, option 4:

nodename

Alias for the node to be cataloged.

hostname

Host name or IPv4 address or IPv6 address 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.

REXX API syntax, option 5:

```
CATALOG TCP/IP6 NODE nodename REMOTE hostname SERVER servicename  
[WITH comment]
```

REXX API parameters, option 5:

nodename

Alias for the node to be cataloged.

hostname

Host name or IPv4 address or IPv6 address 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.

Related reference:

- “sqlencls - End a node directory scan” on page 457
- “sqlengne - Get the next node directory entry” on page 458
- “sqlenops - Start a node directory scan” on page 460
- “sqleuncn - Uncatalog an entry from the node directory” on page 480
- “SQLCA ” on page 526
- “sqle_node_cplic ” on page 541
- “sqle_node_ipxspx ” on page 542
- “sqle_node_local ” on page 543
- “sqle_node_netb ” on page 544
- “sqle_node_npipe ” on page 545
- “sqle_node_struct ” on page 546
- “sqle_node_tcpip ” on page 548

Related samples:

- “nodecat.cbl -- Get node directory information (IBM COBOL)”

sqlectnd - Catalog an entry in the node directory

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

sqlcdcgd - Change a database comment in the system or local database directory

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlcdcgd (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    struct sqlca * pSqlca);
```

```
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);
```

sqlcdcgd API parameters:

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.

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.

sqlcdcgd - Change a database comment in the system or local database directory

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.

pSqlca

Output. A pointer to the sqlca structure.

sqlgcdcgd API-specific 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.

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.

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.

Related reference:

sqlcdc - Change a database comment in the system or local database directory

- “CHANGE DATABASE COMMENT command” in *Command Reference*
- “db2DbDirCloseScan - End a system or local database directory scan” on page 69
- “db2DbDirGetNextEntry - Get the next system or local database directory entry” on page 71
- “db2DbDirOpenScan - Start a system or local database directory scan” on page 75
- “sqlcddb - Catalog a database in the system database directory” on page 400

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++)”

sqlDetachFromCtx - 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlDetachFromCtx (
    void * pCtx,
    void * reserved,
    struct sqlca * pSqlca);
```

sqlDetachFromCtx API parameters:

pCtx Input. A valid context previously allocated by sqlBeginCtx.

reserved

Reserved for future use. Must be set to NULL.

pSqlca

Output. A pointer to the sqlca structure.

Related reference:

- “sqlBeginCtx - Create and attach to an application context” on page 398
- “SQLCA ” on page 526
- “sqlAttachToCtx - Attach to context” on page 397
- “sqlEndCtx - Detach from and free the memory associated with an application context” on page 430
- “sqlGetCurrentCtx - Get current context” on page 447
- “sqlInterruptCtx - Interrupt context” on page 450
- “sqlSetTypeCtx - Set application context type” on page 476

Related samples:

- “dbthrds.sqlc -- How to use multiple context APIs on UNIX (C)”
- “dbthrds.sqlc -- How to use multiple context APIs on UNIX (C++)”

sqledpan - Drop a database on a database partition server

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqledpan (
    char * pDbAlias,
    void * pReserved,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgdpan (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    void * pReserved2,
    char * pDbAlias);
```

sqledpan API parameters:

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.

pReserved

Reserved. Should be NULL.

pSqlca

Output. A pointer to the sqlca structure.

sqlgdpan API-specific parameters:

Reserved1

Reserved for future use.

DbAliasLen

Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

sqledpan - Drop a database on a database partition server

pReserved2

A spare pointer that is set to null or points to zero. Reserved for future use.

Usage notes:

Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*
- “Dropping a database” in *Administration Guide: Implementation*

Related reference:

- “DROP DATABASE command” in *Command Reference*
- “sqlecran - Create a database on a database partition server” on page 406
- “sqledrpd - Drop database” on page 425
- “SQLCA ” on page 526

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqledrpd (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pReserved2,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgdrpd (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pReserved2,
    _SQLOLDCHAR * pDbAlias);
```

sqledrpd API parameters:

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.

pReserved2

A spare pointer that is set to null or points to zero. Reserved for future use.

pSqlca

Output. A pointer to the sqlca structure.

sqlgdrpd API-specific parameters:

Reserved1

Reserved for future use.

sqledrpd - Drop database

DbAliasLen

Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes:

The sqledrpd API 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.

REXX API syntax:

```
DROP DATABASE dbalias
```

REXX API parameters:

dbalias

The alias of the database to be dropped.

Related tasks:

- "Dropping a database" in *Administration Guide: Implementation*

Related reference:

- "DROP DATABASE command" in *Command Reference*
- "sqlcrea - Create database" on page 408
- "sqledpan - Drop a database on a database partition server" on page 423
- "sqleuncd - Uncatalog a database from the system database directory" on page 478
- "SQLCA " on page 526

Related samples:

- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbcreate.C -- Create and drop databases (C++)"

sqlcdrpn - Check whether a database partition server can be dropped

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlcdrpn (
    unsigned short Action,
    void * pReserved,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgdrpn (
    unsigned short Reserved1,
    struct sqlca * pSqlca,
    void * pReserved2,
    unsigned short Action);
```

sqlcdrpn API parameters:

Action

The action requested. The valid value is: SQL_DROPNODE_VERIFY

pReserved

Reserved. Should be NULL.

pSqlca

Output. A pointer to the sqlca structure.

sqlgdrpn API-specific parameters:

Reserved1

Reserved for the length of pReserved2.

pReserved2

A spare pointer that is set to NULL or points to 0. Reserved for future use.

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.

sqlldrpn - Check whether a database partition server can be dropped

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.
2. 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 sqludrdr API or the ALTER DATABASE PARTITION GROUP statement.
3. Drop any event monitors that are defined on the database partition server.
4. Rerun sqlldrpn to ensure that the database partition at the database partition server is no longer in use.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- “Calling DB2 APIs in REXX” on page 754

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*
- “Dropping a database partition” in *Administration Guide: Implementation*

Related reference:

- “sqlleaddn - Add a database partition server to the partitioned database environment” on page 388
- “SQLCA ” on page 526
- “ALTER DATABASE PARTITION GROUP statement” in *SQL Reference, Volume 2*
- “DROP DBPARTITIONNUM VERIFY command” in *Command Reference*

sqledtin - Detach from instance

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqledtin (
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgdtin (
    struct sqlca * pSqlca);
```

sqledtin API parameters:**pSqlca**

Output. A pointer to the sqlca structure.

REXX API syntax:

DETACH

Related tasks:

- “Attaching to and detaching from a non-default instance of the database manager” in *Administration Guide: Implementation*

Related reference:

- “DETACH command” in *Command Reference*
- “sqleatin - Attach to instance” on page 394
- “SQLCA ” on page 526
- “sqleatcp - Attach to instance and change password” on page 391

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++)”

sqlEndCtx - Detach from and free the memory associated with an 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlEndCtx (
    void ** ppCtx,
    sqlint32 IOptions,
    void * reserved,
    struct sqlca * pSqlca);
```

sqlEndCtx API parameters:

ppCtx Output. A data area in private memory (used for the storage of context information) that is freed.

IOptions

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 sqlBeginCtx.

SQL_CTX_END_ALL

If necessary, a call to sqlDetachFromCtx 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 sqlBeginCtx. A call to sqlGetCurrentCtx will be made, and the current context freed from there.

reserved

Reserved for future use. Must be set to NULL.

pSqlca

Output. A pointer to the sqlca structure.

sqlEndCtx - Detach from and free the memory associated with an application context

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 sqlEndCtx.

Related reference:

- “db2CfgGet - Get the database manager or database configuration parameters” on page 49
- “sqlBeginCtx - Create and attach to an application context” on page 398
- “sqledtin - Detach from instance” on page 429
- “sqlDetachFromCtx - Detach from context” on page 422
- “sqlGetCurrentCtx - Get current context” on page 447
- “SQLCA ” on page 526
- “sqlAttachToCtx - Attach to context” on page 397
- “sqlInterruptCtx - Interrupt context” on page 450
- “sqlSetTypeCtx - Set application context type” on page 476

sqlfmem - Free the memory allocated by the sqlbtcq and sqlbmtsq API

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlfmem (
    struct sqlca * pSqlca,
    void * pBuffer);
```

```
SQL_API_RC SQL_API_FN
sqlgfmem (
    struct sqlca * pSqlca,
    void * pBuffer);
```

sqlfmem API parameters:

pSqlca

Output. A pointer to the sqlca structure

pBuffer

Input. Pointer to the memory to be freed.

Related reference:

- "sqlbmtsq - Get the query data for all table spaces" on page 369
- "sqlbtcq - Get the query data for all table space containers" on page 379
- "SQLCA " on page 526

Related samples:

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

sqlfrce - Force users and applications off the system

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 (a database restore, 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
- sysmaint

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlfrce (
    sqlint32 NumAgentIds,
    sqluint32 * pAgentIds,
    unsigned short ForceMode,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgfrce (
    struct sqlca * pSqlca,
    unsigned short ForceMode,
    sqluint32 * pAgentIds,
    sqlint32 NumAgentIds);
```

sqlfrce API parameters:

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 applications with either database connections or instance attachments are forced. If it is set to zero, an error is returned.

sqlfrce - Force users and applications off the system

pAgentIds

Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding database user.

ForceMode

Input. An integer specifying the operating mode of the sqlfrce API. Only the asynchronous mode is supported. This means that the API 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_ASYNC (defined in sqlenv).

pSqlca

Output. A pointer to the sqlca structure.

Usage notes:

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 forced off.

After a force command 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_ASYNC (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 forced off, a unit of work 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 sqlfrce 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.

REXX API syntax:

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API parameters:

ALL All applications will be disconnected. This includes applications that have database connections and applications that have instance attachments.

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 sqlfrce does not wait until all specified applications are terminated before returning.

Related reference:

- “FORCE APPLICATION command” in *Command Reference*
- “FORCE APPLICATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “sqlleatin - Attach to instance” on page 394
- “sqledtin - Detach from instance” on page 429
- “SQLCA ” on page 526

Related samples:

- “dbstop.cbl -- How to stop a database manager (IBM COBOL)”
- “dbsample.sqc -- Creates a sample database (C)”
- “dbconn.sqc -- How to connect to and disconnect from a 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++)”

sqlgddad - Catalog a database in the database connection services (DCS) directory

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgddad (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggddad (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
```

sqlgddad API parameters:

pDCSDirEntry

Input. A pointer to an sql_dir_entry (Database Connection Services directory) structure.

pSqlca

Output. A pointer to the sqlca structure.

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.

sqlgdad - Catalog a database in the database connection services (DCS) directory

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.

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.

Related reference:

- "CATALOG DCS DATABASE command" in *Command Reference*
- "sqlgdcl - End a database connection services (DCS) directory scan" on page 438
- "sqlgdcl - Uncatalog a database from the database connection services (DCS) directory" on page 439
- "sqlgdge - Get a specific entry in the database connection services (DCS) directory" on page 441
- "sqlgdgt - Get database connection services (DCS) directory entries" on page 443
- "sqlgdsc - Start a database connection services (DCS) directory scan" on page 445
- "sql_dir_entry " on page 515
- "SQLCA " on page 526

Related samples:

- "dscat.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++)"

sqlgdcl - End a database connection services (DCS) directory scan

Frees the resources that are allocated by the sqlgdsc API.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgdcl (
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggdcl (
    struct sqlca * pSqlca);
```

sqlgdcl API parameters:

pSqlca

Output. A pointer to the sqlca structure.

REXX API syntax:

```
CLOSE DCS DIRECTORY
```

Related reference:

- "sqlgdgt - Get database connection services (DCS) directory entries" on page 443
- "sqlgdsc - Start a database connection services (DCS) directory scan" on page 445
- "SQLCA " on page 526

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++)"

sqlgedel - Uncatalog a database from the database connection services (DCS) directory

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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgedel (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggdel (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
```

sqlgedel API parameters:

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.

pSqlca

Output. A pointer to the `sqlca` structure.

Usage notes:

DCS databases are also cataloged in the system database directory as remote databases that can be uncataloged using the `sqlleuncd` API.

To recatalog a database in the DCS directory, use the `sqlgedad` API.

To list the DCS databases that are cataloged on a node, use the `sqlgedsc`, `sqlgedgt`, and `sqlgedcl` 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

sqlgdel - Uncatalog a database from the database connection services (DCS) directory

restart (db2start) the database manager. To refresh the directory cache for another application, stop and then restart that application.

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

Related reference:

- “UNCATALOG DCS DATABASE command” in *Command Reference*
- “sqlgdcl - End a database connection services (DCS) directory scan” on page 438
- “sqlgdge - Get a specific entry in the database connection services (DCS) directory” on page 441
- “sqlgdgt - Get database connection services (DCS) directory entries” on page 443
- “sql_dir_entry ” on page 515
- “sqlgdad - Catalog a database in the database connection services (DCS) directory” on page 436
- “sqlgdsc - Start a database connection services (DCS) directory scan” on page 445
- “sqluncd - Uncatalog a database from the system database directory” on page 478
- “SQLCA ” on page 526

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++)”

sqllegdge - Get a specific entry in the database connection services (DCS) directory

Returns information for a specific entry in the Database Connection Services (DCS) directory.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqllegdge (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggdge (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
```

sqllegdge API parameters:

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.

pSqlca

Output. A pointer to the sqlca structure.

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

sqlgedge - Get a specific entry in the database connection services (DCS) directory

XXX.5 PARMS

XXX.6 COMMENT

XXX.7 RESERVED.

Related reference:

- “sqlgedad - Catalog a database in the database connection services (DCS) directory” on page 436
- “sqlgedcl - End a database connection services (DCS) directory scan” on page 438
- “sqlgedgt - Get database connection services (DCS) directory entries” on page 443
- “sqlgedsc - Start a database connection services (DCS) directory scan” on page 445
- “sqlgedel - Uncatalog a database from the database connection services (DCS) directory” on page 439
- “sql_dir_entry ” on page 515

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++)”

sqllegdgt - Get database connection services (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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqllegdgt (
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlggdgt (
    struct sqlca * pSqlca,
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries);
```

sqllegdgt API parameters:

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.

pSqlca

Output. A pointer to the sqlca structure.

Usage notes:

The sqllegdsc API, 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.

REXX API syntax:

```
GET DCS DIRECTORY ENTRY [USING :value]
```

sqlagdgt - Get database connection services (DCS) directory entries

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

Related reference:

- "LIST DCS DIRECTORY command" in *Command Reference*
- "sqlagdcl - End a database connection services (DCS) directory scan" on page 438
- "sqlagdge - Get a specific entry in the database connection services (DCS) directory" on page 441
- "sqlagdsc - Start a database connection services (DCS) directory scan" on page 445
- "sql_dir_entry " on page 515
- "SQLCA " on page 526

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++)"

sqllegdsc - Start a database connection services (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 sqllegdgt API and sqllegdcl API to release the resources associated with calling this API.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqllegdsc (
    short * pNumEntries,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggdsc (
    struct sqlca * pSqlca,
    short * pNumEntries);
```

sqllegdsc API parameters:

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.

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.

REXX API syntax:

```
OPEN DCS DIRECTORY
```

Related reference:

- “sqllegdcl - End a database connection services (DCS) directory scan” on page 438
- “sqllegdge - Get a specific entry in the database connection services (DCS) directory” on page 441

sqlegdsc - Start a database connection services (DCS) directory scan

- “sqlegdgt - Get database connection services (DCS) directory entries” on page 443
- “SQLCA ” on page 526

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++)”

sqlGetCurrentCtx - 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlGetCurrentCtx (
    void ** ppCtx,
    void * reserved,
    struct sqlca * pSqlca);
```

sqlGetCurrentCtx API parameters:

ppCtx Output. A data area allocated out of private memory for the storage of context information.

reserved

Reserved for future use. Must be set to NULL.

pSqlca

Output. A pointer to the sqlca structure.

Related reference:

- “SQLCA ” on page 526
- “sqlAttachToCtx - Attach to context” on page 397
- “sqlBeginCtx - Create and attach to an application context” on page 398
- “sqlDetachFromCtx - Detach from context” on page 422
- “sqlEndCtx - Detach from and free the memory associated with an application context” on page 430
- “sqlInterruptCtx - Interrupt context” on page 450
- “sqlSetTypeCtx - Set application context type” on page 476

sqlgins - Get current instance

Returns the value of the DB2INSTANCE environment variable.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgins (
    _SQLOLDCHAR * pInstance,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggins (
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pInstance);
```

sqlgins API parameters:

pInstance

Output. Pointer to a string buffer where the database manager instance name is placed. This buffer must be at least 9 bytes in length, including 1 byte for the null terminating character.

pSqlca

Output. A pointer to the sqlca structure.

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 `sqlcat in - Attach`, with null arguments except for the sqlca structure.

REXX API syntax:

```
GET INSTANCE INTO :instance
```

REXX API parameters:

instance

A REXX host variable into which the database manager instance name is to be placed.

Related tasks:

- "Setting the current instance environment variables" in *Administration Guide: Implementation*

Related reference:

- “GET INSTANCE command” in *Command Reference*
- “sqlcatin - Attach to instance” on page 394
- “SQLCA ” on page 526
- “ENV_INST_INFO administrative view – Retrieve information about the current instance” in *Administrative SQL Routines and Views*

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++)”

sqlInterruptCtx - 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlInterruptCtx (
    void * pCtx,
    void * reserved,
    struct sqlca * pSqlca);
```

sqlInterruptCtx API parameters:

pCtx Input. A valid context previously allocated by sqlBeginCtx.

reserved

Reserved for future use. Must be set to NULL.

pSqlca

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:

- “sqlBeginCtx - Create and attach to an application context” on page 398
- “SQLCA ” on page 526
- “sqlAttachToCtx - Attach to context” on page 397
- “sqlDetachFromCtx - Detach from context” on page 422
- “sqlEndCtx - Detach from and free the memory associated with an application context” on page 430
- “sqlGetCurrentCtx - Get current context” on page 447
- “sqlSetTypeCtx - Set application context type” on page 476

sqlintr - Interrupt application requests

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 `sqlsig - 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`

API and data structure syntax:

```
SQL_API_RC SQL_API_INTR
    sqlintr (void);
```

```
SQL_API_RC SQL_API_FN
    sqlgintr (
        void);
```

sqlintr API parameters:

None

Usage notes:

No database manager APIs should be called from an interrupt handler except `sqlintr`. 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 23. INTERRUPT actions

Database activity	Action
BACKUP	Utility cancelled. Data on media may be incomplete.
BIND	Binding cancelled. Package creation rolled back.
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.

sqleintr - Interrupt application requests

Table 23. INTERRUPT actions (continued)

Database activity	Action
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.

REXX API syntax:

INTERRUPT

Examples:

```
call SQLDBS 'INTERRUPT'
```

Related reference:

- “sqleisig - Install signal handler” on page 453

sqlsig - 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 `sqlintr`.

Authorization:

None

Required connection:

None

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlsig (
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlsig (
    struct sqlca * pSqlca);
```

sqlsig API parameters:

pSqlca

Output. A pointer to the `sqlca` structure.

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 `sqlintr` API can be developed. Use either the operating system call or the language-specific library signal function. The `sqlintr` 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.

REXX API syntax:

```
INSTALL SIGNAL HANDLER
```

Related reference:

- "sqlintr - Interrupt application requests" on page 451
- "SQLCA " on page 526

Related samples:

sqleisig - Install signal handler

- "dbcmt.cbl -- Change a database comment in the database directory (IBM COBOL)"

sqlmgdb - Migrate previous version of DB2 database to current version

Converts previous (Version 8.x or higher) versions of DB2 databases to current versions.

Authorization:

sysadm

Required connection:

This API establishes a database connection.

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlmgdb (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pPassword,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlmgdb (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pPassword,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pDbAlias);
```

sqlmgdb API parameters:

pDbAlias

Input. A string containing the alias of the database that is cataloged in the system database directory.

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.

pSqlca

Output. A pointer to the sqlca structure.

sqlmgdb API-specific 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.

sqlmgdb - Migrate previous version of DB2 database to current version

DbAliasLen

Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes:

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.

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.

Related reference:

- "MIGRATE DATABASE command" in *Command Reference*
- "SQLCA " on page 526

Related samples:

- "migrate.cbl -- Demonstrates how to migrate to a database (IBM COBOL)"
- "dbmigrat.c -- Migrate a database (C)"
- "dbmigrat.C -- Migrate a database (C++)"

sqlencs - End a node directory scan

Frees the resources that are allocated by the sqlenops API.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlencs (
    unsigned short Handle,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgncs (
    unsigned short Handle,
    struct sqlca * pSqlca);
```

sqlencs 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 the next node directory entry” on page 458
- “sqlenops - Start a node directory scan” on page 460
- “SQLCA ” on page 526

Related samples:

- “nodecat.cbl -- Get node directory information (IBM COBOL)”
- “ininfo.c -- Set and get information at the instance level (C)”
- “ininfo.C -- Set and get information at the instance level (C++)”

sqlengne - Get the 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

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlengne (
    unsigned short Handle,
    struct sqleninfo ** ppNodeDirEntry,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgngne (
    unsigned short Handle,
    struct sqleninfo ** ppNodeDirEntry,
    struct sqlca * pSqlca);
```

sqlengne 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.

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.

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).

Related reference:

- "LIST NODE DIRECTORY command" in *Command Reference*
- "sqlencls - End a node directory scan" on page 457
- "sqlenops - Start a node directory scan" on page 460
- "SQLCA " on page 526
- "sqleninfo " on page 564

Related samples:

- "nodecat.cbl -- Get node directory information (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlenops - Start a 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.

Call the sqlengne API to advance through the node directory and examine information about the node entries. Close the scan by calling the sqlencls API. This removes the copy of the directory from memory.

Authorization:

None

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlenops (
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgnops (
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
```

sqlenops API parameters:

pHandle

Output. Identifier returned from this API. This identifier must be passed to the sqlengne API and sqlencls API.

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.

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.

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.

Related reference:

- “sqlencls - End a node directory scan” on page 457
- “sqlengne - Get the next node directory entry” on page 458
- “SQLCA ” on page 526

Related samples:

- “nodecat.cbl -- Get node directory information (IBM COBOL)”
- “ininfo.c -- Set and get information at the instance level (C)”
- “ininfo.C -- Set and get information at the instance level (C++)”

sqleqryc - Query client connection settings

Returns current connection settings for an application process. The `sqle_conn_setting` data structure is populated with the connection setting types and values.

Authorization:

None

Required connection:

None

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqleqryc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgqryc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
```

sqleqryc 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.

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.

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. The SYNCPOINT option is ignored and is available only for backward compatibility. 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.

Related reference:

- “QUERY CLIENT command” in *Command Reference*
- “sqleqryi - Query client information” on page 464
- “sqlesetc - Set client connection settings” on page 470
- “SQLCA ” on page 526
- “sqle_conn_setting ” on page 538

Related samples:

- “client.cbl -- How to set and query a client (IBM COBOL)”
- “cli_info.c -- Set and get information at the client level (C)”
- “cli_info.C -- Set and get information at the client level (C++)”

sqleqryi - Query client information

Returns existing client information by populating the fields in the `sqle_client_info` data structure. 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`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqleqryi (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info* pClient_Info,
    struct sqlca * pSqlca);
```

sqleqryi 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 473
- “SQLCA ” on page 526
- “sqle_client_info ” on page 535
- “SQLGetConnectAttr function (CLI) - Get current attribute setting” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value” in *Call Level Interface Guide and Reference, Volume 2*
- “sqleqryc - Query client connection settings” on page 462

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++)”

sqlsact - 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlsact (
    char * pAccountingString,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgsact (
    unsigned short AccountingStringLen,
    char * pAccountingString,
    struct sqlca * pSqlca);
```

sqlsact API parameters:

pAccountingString

Input. A string containing the accounting data.

pSqlca

Output. A pointer to the sqlca structure.

sqlgsact API-specific parameters:

AccountingStringLen

Input. A 2-byte unsigned integer representing the length in bytes of the accounting string.

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 473
- "SQLCA " on page 526

Related samples:

- "setact.cbl -- How to set accounting string (IBM COBOL)"

sqlesdeg - Set the maximum runtime intra-partition parallelism level or degree for SQL statements

Sets the maximum run time degree of intra-partition parallelism for SQL statement execution for specified active applications. It has no effect on CREATE INDEX statement execution 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlesdeg (
    sqlint32 NumAgentIds,
    sqluint32 * pAgentIds,
    sqlint32 Degree,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgsdeg (
    struct sqlca * pSqlca,
    sqlint32 Degree,
    sqluint32 * pAgentIds,
    sqlint32 NumAgentIds);
```

sqlesdeg API parameters:

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.

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.

sqlsdeg - Set the maximum runtime intra-partition parallelism level or degree for SQL statements

Degree

Input. The new value for the maximum run time degree of parallelism. The value must be in the range 1 to 32767.

pSqlca

Output. A pointer to the sqlca structure.

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.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*

Related reference:

- “SET RUNTIME DEGREE command” in *Command Reference*
- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “SQLCA ” on page 526

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 connection settings

Specifies connection settings for the application. Use the `sqle_conn_setting` data structure to specify the connection setting types and values.

Authorization:

None

Required connection:

None

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlesetc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgsetc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
```

sqlesetc 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.

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.

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. The SYNCPOINT option is ignored and is available only for backward compatibility.

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

Related reference:

- "SET CLIENT command" in *Command Reference*

sqlesetc - Set client connection settings

- “sqleqryc - Query client connection settings” on page 462
- “sqleseti - Set client information” on page 473
- “SQLCA ” on page 526
- “sqle_conn_setting ” on page 538

Related samples:

- “client.cbl -- How to set and query a client (IBM COBOL)”
- “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++)”
- “cli_info.C -- Set and get information at the client level (C++)”

sqlseti - Set client information

Permits an application to set client information (by setting the fields in the `sql_client_info` data structure) 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`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlseti (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sql_client_info* pClient_Info,
    struct sqlca * pSqlca);
```

sqlseti 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.

sqleseti - Set client information

NumItems

Input. Number of entries being modified. The minimum value is 1.

pClient_Info

Input. A pointer to an array of NumItems `sqlc_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 `sqlcqryi` API.

Calling the `sqleseti` API in a CLI program before a connection will not work. Calling the `sqleseti` API in a CLI program after a connection has been established may result in unpredictable behavior. It is recommended that the corresponding CLI functions `SQLSetConnectAttr()` or `SQLSetEnvAttr()` be used instead.

Related reference:

- “`SQLSetConnectAttr` function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*
- “`SQLSetEnvAttr` function (CLI) - Set environment attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “`sqlc_client_info`” on page 535
- “`sqlcqryi` - Query client information” on page 464
- “`sqlsact` - Set accounting string” on page 466

- “sqlesei - Set client connection settings” on page 470
- “SQLCA ” on page 526

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++)”

sqlSetTypeCtx - 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlSetTypeCtx (
    sqlint32 IOptions);
```

sqlSetTypeCtx API parameters:

IOptions

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

- sqlBeginCtx API
- sqlAttachToCtx API
- sqlDetachFromCtx API
- sqlEndCtx API

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.
- sqlintr API interrupts all contexts. To interrupt a specific context, use sqlInterruptCtx.

Usage notes:

This API must be called before any other database call, and only the first call is effective.

Related reference:

- “sqlDetachFromCtx - Detach from context” on page 422

sqlcSetTypeCtx - Set application context type

- “sqlcEndCtx - Detach from and free the memory associated with an application context” on page 430
- “sqlcAttachToCtx - Attach to context” on page 397
- “sqlcBeginCtx - Create and attach to an application context” on page 398
- “sqlcintr - Interrupt application requests” on page 451
- “sqlcInterruptCtx - Interrupt context” on page 450
- “sqlcGetCurrentCtx - Get current context” on page 447

Related samples:

- “dbthrds.sqlc -- How to use multiple context APIs on UNIX (C)”
- “dbthrds.sqlC -- How to use multiple context APIs on UNIX (C++)”

sqlenv - Uncatalog a database from the system database directory

Deletes an entry from the system database directory.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlenv (
    _SQLDCHAR * pDbAlias,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlguncd (
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLDCHAR * pDbAlias);
```

sqlenv API parameters:

pDbAlias

Input. A string containing the database alias that is to be uncataloged.

pSqlca

Output. A pointer to the sqlca structure.

sqlguncd API-specific parameters:

DbAliasLen

Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes:

Only entries in the system database directory can be uncataloged. Entries in the local database directory can be deleted using the sqldrpd API.

To recatalog the database, use the sqlcadb API.

To list the databases that are cataloged on a node, use the db2DbDirOpenScan, db2DbDirGetNextEntry, and db2DbDirCloseScan 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.

sqlleuncd - Uncatalog a database from the system database directory

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.

REXX API syntax:

```
UNCATALOG DATABASE dbname
```

REXX API parameters:

dbname

Alias of the database to be uncataloged.

Related reference:

- “UNCATALOG DATABASE command” in *Command Reference*
- “db2DbDirCloseScan - End a system or local database directory scan” on page 69
- “db2DbDirGetNextEntry - Get the next system or local database directory entry” on page 71
- “db2DbDirOpenScan - Start a system or local database directory scan” on page 75
- “sqlcadb - Catalog a database in the system database directory” on page 400
- “sqledrpd - Drop database” on page 425
- “SQLCA ” on page 526

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++)”

sqluncn - Uncatalog an entry from the node directory

Deletes an entry from the node directory.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqluncn (
    _SQLOLDCHAR * pNodeName,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlguncn (
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pNodeName);
```

sqluncn API parameters:

pNodeName

Input. A string containing the name of the node to be uncataloged.

pSqlca

Output. A pointer to the sqlca structure.

sqlguncn API-specific parameters:

NodeNameLen

Input. A 2-byte unsigned integer representing the length in bytes of the node name.

Usage notes:

To recatalog the node, use the sqlctnd API.

To list the nodes that are cataloged, use the db2DbDirOpenScan, db2DbDirGetNextEntry, and db2DbDirCloseScan 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

sqleuncn - Uncatalog an entry from the node directory

restart (db2start) the database manager. To refresh the directory cache for another application, stop and then restart that application.

REXX API syntax:

```
UNCATALOG NODE nodename
```

REXX API parameters:

nodename

Name of the node to be uncataloged.

Related reference:

- “UNCATALOG NODE command” in *Command Reference*
- “sqlctnd - Catalog an entry in the node directory” on page 415
- “sqlencls - End a node directory scan” on page 457
- “sqlengne - Get the next node directory entry” on page 458
- “sqlenops - Start a node directory scan” on page 460
- “SQLCA ” on page 526

Related samples:

- “nodecat.cbl -- Get node directory information (IBM COBOL)”
- “ininfo.c -- Set and get information at the instance level (C)”
- “ininfo.C -- Set and get information at the instance level (C++)”

sqlgaddr - Get the address of a variable

Places the address of a variable into another variable. This API is used in host programming languages, FORTRAN and COBOL, which do not provide pointer manipulation.

Authorization:

None

Required connection:

None

API include file:

sqlutil.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgaddr (
    char * pVariable,
    char ** ppOutputAddress);
```

sqlgaddr API parameters:

pVariable

Input. Variable whose address is to be returned.

ppOutputAddress

Output. A 4-byte area into which the variable address is returned.

Related reference:

- “sqlgaddr - Dereference an address” on page 483

sqlgdref - Dereference an address

Copies data from a buffer that is defined by a pointer, into a variable that is directly accessible by the application. This API is used in host programming languages, FORTRAN and COBOL, which 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgdref (
    unsigned int NumBytes,
    char * pTargetBuffer,
    char ** ppSourceBuffer);
```

sqlgdref 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.

Related reference:

- “sqlgaddr - Get the address of a variable” on page 482

sqlgmcpy - Copy data from one memory area to another

Copies data from one memory area to another. This API is used in host programming languages, FORTRAN and COBOL, that do not provide memory block copy functions.

Authorization:

None

Required connection:

None

API include file:

sqlutil.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgmcpy (
    void * pTargetBuffer,
    const void * pSource,
    sqluint32 NumBytes);
```

sqlgmcpy 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.

Related reference:

- “sqlgaddr - Get the address of a variable” on page 482

sqlgostt - Get the SQLSTATE message

Retrieves the message text associated with an SQLSTATE value.

Authorization:

None

Required connection:

None

API include file:

sql.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlgostt (
    char * pBuffer,
    short BufferSize,
    short LineWidth,
    char * pSqlstate);
```

```
SQL_API_RC SQL_API_FN
sqlggstt (
    short BufferSize,
    short LineWidth,
    char * pSqlstate,
    char * pBuffer);
```

sqlgostt API parameters:

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.

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.

Usage notes:

One message is returned per call.

A LF/NULL sequence is placed at the end of each message.

sqlgostt - Get the SQLSTATE 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.

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.

Related reference:

- “sqlaintp - Get error message” on page 353

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++)”

sqluadai - Get current user's authorities

Reports the instance level and database level authorities of the current user from values found in the database manager configuration file and the authorization system catalog view (SYSCAT.DBAUTH) respectively. The instance level authorities reported are the ones that you can set in the `sysadm_group`, `sysmaint_group` and `sysctrl_group` database manager configuration parameters and the database level authorities are the ones that can be granted via the GRANT (Database Authorities) statement.

Authorization:

None

Required connection:

Database

API include file:

sqlutil.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqluadai (
    struct sql_authorizations * pAuthorizations,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgadai (
    struct sql_authorizations * pAuthorizations,
    struct sqlca * pSqlca);
```

sqluadai API parameters:**pAuthorizations**

Input or 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.

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.

REXX API syntax:

sqluadau - Get current user's authorities

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.

Related reference:

- "sql_authorizations " on page 513
- "SQLCA " on page 526

sqlldrdt - Redistribute data across a 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 data basepartitions 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlldrdt (
    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);
```

```
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);
```

sqlldrdt API parameters:

pNodeGroupName

The name of the database partition group to be redistributed.

sqlldr - Redistribute data across a database partition group

pTargetPMapFileName

The name of the file that contains the target distribution 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 database 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 database partition represents the same amount of data). If pDataDistFileName parameter is not null, the values in this file are assumed to represent the current data distribution. When the DataRedistOption is U, the pTargetPMapFileName parameter 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 the pTargetPMapFileName parameter. For this option, the parameters, pDataDistFileName, pAddList, and pDropList should be null, and both the parameters, AddCount and DropCount must be zero.
- C** Specifies to continue a redistribution operation that failed. For this option, the parameters, pTargetPMapFileName, pDataDistFileName, pAddList, and pDropList should be null, and both the parameters, AddCount and DropCount must be zero.
- R** Specifies to roll back a redistribution operation that failed. For this

sqludrdt - Redistribute data across a database partition group

option, the parameters, pTargetPMapFileName, pDataDistFileName, pAddList, and pDropList should be null, and both the parameters, AddCount and DropCount must be zero.

pSqlca

Output. A pointer to the sqlca structure.

sqlgdrdt API-specific parameters:

NodeGroupNameLen

The length of the name of the database partition group.

TargetPMapFileNameLen

The length of the name of the target distribution map file.

DataDistFileNameLen

The length of the name of the data distribution file.

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 the DATA CAPTURE CHANGES clause cannot be redistributed.

Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables in the database partition group.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

sqludrtd - Redistribute data across a database partition group

- “Calling DB2 APIs in REXX” on page 754

Related tasks:

- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*
- “Redistributing data across database partitions” in *Performance Guide*

Related reference:

- “db2Runstats - Update statistics for tables and indexes” on page 306
- “sqlarbind - Rebind package” on page 358
- “SQLCA ” on page 526
- “REDISTRIBUTE DATABASE PARTITION GROUP command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “ALTER DATABASE PARTITION GROUP statement” in *SQL Reference, Volume 2*
- “LIST DATABASE DIRECTORY command” in *Command Reference*
- “REDISTRIBUTE DATABASE PARTITION GROUP command” in *Command Reference*

sqlugrpn - Get the database partition server number for a row

Returns the database partition number and the database partition server number based on the distribution key values. An application can use this information to determine on 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 distribution key values. The output is a database partition number generated by the distribution strategy and the corresponding database partition server number from the distribution map. If the distribution map information is not provided, only the database 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 must be invoked from a database partition server in the `db2nodes.cfg` file. This API should not be invoked from a client, since it could result in erroneous database partitioning information being returned due to differences in codepage and endianness between the client and the server.

Authorization:

None

API include file:

`sqlutil.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlugrpn (
    unsigned short num_ptrs,
    unsigned char ** ptr_array,
    unsigned short * ptr_lens,
    unsigned short territory_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);
```

```
SQL_API_RC SQL_API_FN
sqlggrpn (
    unsigned short num_ptrs,
    unsigned char ** ptr_array,
    unsigned short * ptr_lens,
    unsigned short territory_code,
    unsigned short codepage,
    struct sqlupi * part_info,
    short * part_num,
    SQL_PDB_NODE_TYPE * node_num,
    unsigned short chklvl,
```

sqlugrpn - Get the database partition server number for a row

```
struct sqlca * sqlca,  
short dataformat,  
void * pReserved1,  
void * pReserved2);
```

sqlugrpn API parameters:

num_ptrs

The number of pointers in ptr_array. The value must be the same as the one specified for the part_info parameter; 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 distribution 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.

territory_ctype

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 database 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 distribution key values. Valid values are:

SQL_CHARSTRING_FORMAT

All distribution 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.

sqlugrpn - Get the database partition server number for a row

SQL_PACKEDDECIMAL_FORMAT

All decimal column distribution key values are in packed decimal format.

SQL_BINARYNUMERICS_FORMAT

All numeric distribution 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 distribution key.

Note: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types must be converted to the database 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 distribution map must be supplied; that is, pmaplen field in part_info parameter (part_info->pmaplen) is either 2 or 8192. Otherwise, SQLCODE -6038 is returned. The distribution key must be defined; that is, sqld field in part_info parameter (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:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "sqlugtpi - Get table distribution information" on page 496
- "sqludrtd - Redistribute data across a database partition group" on page 489
- "SQLCA " on page 526
- "sqlupi " on page 588
- "db2CfgGet - Get the database manager or database configuration parameters" on page 49

sqlugtpi - Get table distribution information

Allows an application to obtain the distribution information for a table. The distribution information includes the distribution map and the column definitions of the distribution key. Information returned by this API can be passed to the sqlugrpn API to determine the database 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 distribution 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqlugtpi (
    unsigned char * tablename,
    struct sqlupi * part_info,
    struct sqlca * sqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggtpi (
    unsigned short tn_length,
    unsigned char * tablename,
    struct sqlupi * part_info,
    struct sqlca * sqlca);
```

sqlugtpi API parameters:

tablename

The fully qualified name of the table.

part_info

A pointer to the sqlupi structure.

pSqlca

Output. A pointer to the sqlca structure.

sqlggtpi API-specific parameters:

tn_length

A 2-byte unsigned integer with the length of the table name.

Related reference:

- “sqludrtd - Redistribute data across a database partition group” on page 489
- “sqlugrpn - Get the database partition server number for a row” on page 493
- “SQLCA ” on page 526
- “sqlupi ” on page 588

sqluvqdp - Quiesce table spaces for a 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
- 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

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
sqluvqdp (
    char * pTableName,
    sqlint32 QuiesceMode,
    void * pReserved,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlgvqdp (
    unsigned short TableNameLen,
    char * pTableName,
    sqlint32 QuiesceMode,
    void * pReserved,
    struct sqlca * pSqlca);
```

sqluvqdp API parameters:

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.

sqluvqdp - Quiesce table spaces for a table

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.

sqlgvqdp API-specific parameters:

TableNameLen

Input. A 2-byte unsigned integer representing the length in bytes of the table name.

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

sqluvqdp - Quiesce table spaces for a table

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`.

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.

Related reference:

- “QUIESCE TABLESPACES FOR TABLE command” in *Command Reference*
- “db2DatabaseQuiesce - Quiesce the database” on page 62
- “db2InstanceQuiesce - Quiesce instance” on page 170
- “SQLCA ” on page 526
- “QUIESCE TABLESPACES FOR TABLE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

Related samples:

- “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++)”

sqlxhfrg - Forget transaction status

Permits the resource manager to release resources held by a heuristically completed transaction (that is, one that has been committed or rolled back heuristically). You might call this API after heuristically committing or rolling back an indoubt XA transaction.

Authorization:

None

Required connection:

Database

API include file:

sqlxa.h

API and data structure syntax:

```
extern int SQL_API_FN sqlxhfrg(
    SQLXA_XID *pTransId,
    struct sqlca *pSqlca
);
```

sqlxhfrg 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 concepts:

- “Indoubt transaction management APIs” in *Administration Guide: Planning*

Related reference:

- “LIST DRDA INDOUBT TRANSACTIONS command” in *Command Reference*
- “LIST INDOUBT TRANSACTIONS command” in *Command Reference*
- “SQLCA ” on page 526
- “SQLXA_XID ” on page 590
- “sqlxphcm - Commit an indoubt transaction” on page 502
- “db2XaListIndTrans - List indoubt transactions” on page 345
- “sqlxphrl - Roll back an indoubt transaction” on page 504

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:

None

Required connection:

Database

API include file:

sqlxa.h

API and data structure syntax:

```
extern int SQL_API_FN sqlxphcm(  
    int exe_type,  
    SQLXA_XID *pTransId,  
    struct sqlca *pSqlca  
);
```

sqlxphcm 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 the sqlxhfrg API is called.

Related concepts:

- "Indoubt transaction management APIs" in *Administration Guide: Planning*

Related reference:

- "LIST DRDA INDOUBT TRANSACTIONS command" in *Command Reference*
- "LIST INDOUBT TRANSACTIONS command" in *Command Reference*
- "SQLXA_XID " on page 590
- "sqlxhfrg - Forget transaction status" on page 501
- "SQLCA " on page 526

sqlxphcm - Commit an indoubt transaction

- “db2XaListIndTrans - List indoubt transactions” on page 345
- “sqlxphrl - Roll back an indoubt transaction” on page 504

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:

None

Required connection:

Database

API include file:

sqlxa.h

API and data structure syntax:

```
extern int SQL_API_FN sqlxphrl(  
    int exe_type,  
    SQLXA_XID *pTransId,  
    struct sqlca *pSqlca  
);
```

sqlxphrl 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 the sqlxhfrg API is called.

Related concepts:

- "Indoubt transaction management APIs" in *Administration Guide: Planning*

Related reference:

- "LIST DRDA INDOUBT TRANSACTIONS command" in *Command Reference*
- "LIST INDOUBT TRANSACTIONS command" in *Command Reference*
- "sqlxhfrg - Forget transaction status" on page 501
- "SQLCA " on page 526
- "SQLXA_XID " on page 590
- "sqlxphcm - Commit an indoubt transaction" on page 502

sqlxphrl - Roll back an indoubt transaction

- “db2XaListIndTrans - List indoubt transactions” on page 345

sqlxphrl - Roll back an indoubt transaction

Chapter 2. Data structures used by multiple APIs

This section describes the data structures that are used by multiple APIs.

db2HistoryData

This structure is used to return information after a call to the db2HistoryGetEntry API.

Table 24. Fields in the db2HistoryData 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 <code>yyyymmddhhmmss</code> , 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 <code>yyyymmddhhmmss</code> , indicating when the operation was completed.

Table 24. Fields in the db2HistoryData Structure (continued)

Field Name	Data Type	Description
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.
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.

Table 24. Fields in the db2HistoryData Structure (continued)

Field Name	Data Type	Description
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 parameter 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 parameter: - 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.
iNumTablespaces	db2Uint32	Number of entries in the poTablespace list that are available for use by the db2HistoryGetEntry API.

Table 24. Fields in the db2HistoryData Structure (continued)

Field Name	Data Type	Description
oNumTablespaces	db2UInt32	Number of entries in the poTablespace list that were used by the db2HistoryGetEntry API. 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 following table.
oObject	char	Granularity of the operation: D for full database, P for table space, and T for table.
oOptype	char	See the table titled "Valid oOptype Values in the db2HistData Structure".
oStatus	char	Entry status: A for action, D for deleted ,E for expired, I for inactive, N for not yet committed, 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 25. Valid oOperation Values in the db2HistoryData 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

Table 25. Valid oOperation Values in the db2HistoryData Structure (continued)

Value	Description	C Definition	COBOL/FORTRAN Definition
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
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
X	log archive	DB2HISTORY_OP_ARCHIVE_LOG	DB2HIST_OP_ARCHIVE_LOG

Table 26. Valid oOptype Values in the db2HistData Structure

oOperation	oOptype	Description	C/COBOL/FORTRAN Definition
B	F N I O D E	offline, online, incremental offline, incremental online, delta offline, delta online	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE, DB2HISTORY_OPTYPE_INCR_OFFLINE, DB2HISTORY_OPTYPE_INCR_ONLINE, DB2HISTORY_OPTYPE_DELTA_OFFLINE, DB2HISTORY_OPTYPE_DELTA_ONLINE
F	E P	end of logs, point in time	DB2HISTORY_OPTYPE_EOL, DB2HISTORY_OPTYPE_PIT
G	F N	offline, online	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE
L	I R	insert, replace	DB2HISTORY_OPTYPE_INSERT, DB2HISTORY_OPTYPE_REPLACE
Q	S U X Z	quiesce share, quiesce update, quiesce exclusive, quiesce reset	DB2HISTORY_OPTYPE_SHARE, DB2HISTORY_OPTYPE_UPDATE, DB2HISTORY_OPTYPE_EXCL, DB2HISTORY_OPTYPE_RESET
R	F N I O R	offline, online, incremental offline, incremental online, rebuild	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE, DB2HISTORY_OPTYPE_INCR_OFFLINE, DB2HISTORY_OPTYPE_INCR_ONLINE, DB2HISTORY_OPTYPE_REBUILD
T	C R	add containers, rebalance	DB2HISTORY_OPTYPE_ADD_CONT, DB2HISTORY_OPTYPE_REB
X	N P M F 1 2	archive log command, primary log path, mirror log path, archive fail path, log archive method 1, log archive method 2	DB2HISTORY_OPTYPE_ARCHIVE_CMD, DB2HISTORY_OPTYPE_PRIMARY, DB2HISTORY_OPTYPE_MIRROR, DB2HISTORY_OPTYPE_ARCHFAIL, DB2HISTORY_OPTYPE_ARCH1, DB2HISTORY_OPTYPE_ARCH2

Table 27. Fields in the db2HistoryEID Structure

Field Name	Data Type	Description
ioNode ioHID	SQL_PDB_NODE_TYPE db2UInt32	Node number. Local history file entry ID.

db2HistoryData

API and data structure syntax:

```
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;
    struct db2Char *poTablespace;
    db2UInt32 iNumTablespaces;
    db2UInt32 oNumTablespaces;
    char oOperation;
    char oObject;
    char oOptype;
    char oStatus;
    char oDeviceType;
} db2HistoryData;

typedef SQL_STRUCTURE db2Char
{
    char *pioData;
    db2UInt32 iLength;
    db2UInt32 oLength;
} db2Char;

typedef SQL_STRUCTURE db2HistoryEID
{
    SQL_PDB_NODE_TYPE ioNode;
    db2UInt32 ioHID;
} db2HistoryEID;
```

db2Char data structure parameters:

pioData

A pointer to a character data buffer. If NULL, no data will be returned.

iLength

Input. The size of the pioData buffer.

oLength

Output. The number of valid characters of data in the pioData buffer.

db2HistoryEID data structure parameters:

ioNode

This parameter can be used as either an input or output parameter. Indicates the node number.

ioHID This parameter can be used as either an input or output parameter. Indicates the local history file entry ID.

Related reference:

- “db2HistoryGetEntry - Get the next entry in the history file” on page 129
- “SQLCA ” on page 526

sql_authorizations

This structure is used to return information after a call to the `squadau` 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 28. Fields in the `SQL-AUTHORIZATIONS` Structure

Field Name	Description
<code>SQL_AUTHORIZATIONS_LEN</code>	Size of structure. <code>SYSADM</code> authority.
<code>SQL_SYSADM_AUTH</code>	<code>SYSCTRL</code> authority. <code>SYSMAINT</code> authority.
<code>SQL_SYSCTRL_AUTH</code>	<code>DBADM</code> authority. <code>CREATETAB</code> authority.
<code>SQL_SYSMAINT_AUTH</code>	<code>CREATE_NOT_FENCED</code> authority.
<code>SQL_DBADM_AUTH</code>	<code>BINDADD</code> authority. <code>CONNECT</code> authority.
<code>SQL_CREATETAB_AUTH</code>	<code>IMPLICIT_SCHEMA</code> authority. <code>LOAD</code>
<code>SQL_CREATET_NOT_FENC_AUTH</code>	authority. User belongs to a group which
<code>SQL_BINDADD_AUTH</code>	holds <code>SYSADM</code> authority. User belongs to a
<code>SQL_CONNECT_AUTH</code>	group which holds <code>SYSCTRL</code> authority. User
<code>SQL_IMPLICIT_SCHEMA_AUTH</code>	belongs to a group which holds <code>SYSMAINT</code>
<code>SQL_LOAD_AUTH</code>	authority. User belongs to a group which
<code>SQL_SYSADM_GRP_AUTH</code>	holds <code>DBADM</code> authority. User belongs to a
<code>SQL_SYSCTRL_GRP_AUTH</code>	group which holds <code>CREATETAB</code> authority.
<code>SQL_SYSMAINT_GRP_AUTH</code>	User belongs to a group which holds
<code>SQL_DBADM_GRP_AUTH</code>	<code>CREATE_NOT_FENCED</code> authority. User
<code>SQL_CREATETAB_GRP_AUTH</code>	belongs to a group which holds <code>BINDADD</code>
<code>SQL_CREATE_NON_FENC_GRP_AUTH</code>	authority. User belongs to a group which
<code>SQL_BINDADD_GRP_AUTH</code>	holds <code>CONNECT</code> authority. User belongs to
<code>SQL_CONNECT_GRP_AUTH</code>	a group which holds <code>IMPLICIT_SCHEMA</code>
<code>SQL_IMPLICIT_SCHEMA_GRP_AUTH</code>	authority. User belongs to a group which
<code>SQL_LOAD_GRP_AUTH</code>	holds <code>LOAD</code> authority.
<code>SQL_CREATE_EXT_RTN_AUTH</code>	<code>CREATE_EXTERNAL_ROUTINE</code> authority.
<code>SQL_CREATE_EXT_RTN_GRP_AUTH</code>	User belongs to a group which holds
<code>SQL_QUIESCE_CONNECT_AUTH</code>	<code>CREATE_EXTERNAL_ROUTINE</code> authority.
<code>SQL_QUIESCE_CONNECT_GRP_AUTH</code>	<code>QUIESCE CONNECT</code> authority. User
<code>SQL_SECURITY_ADMIN_AUTH</code>	belongs to a group which holds <code>QUIESCE</code>
<code>SQL_SECURITY_ADMIN_GRP_AUTH</code>	<code>CONNECT</code> authority. <code>SECADM</code> authority.
<code>SQL_SYSMON_AUTH</code>	User belongs to a group which holds
<code>SQL_SYSMON_GRP_AUTH</code>	<code>SECADM</code> authority. <code>SYSMON</code> authority.
	User belongs to a group which holds
	<code>SYSMON</code> 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.

API and data structure syntax:

```
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;
```

sql_authorizations

```
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;  
short sql_create_ext_rtn_auth;  
short sql_create_ext_rtn_grp_auth;  
short sql_quiesce_connect_auth;  
short sql_quiesce_connect_grp_auth;  
short sql_security_admin_auth;  
short sql_security_admin_grp_auth;  
short sql_library_admin_auth;  
short sql_library_admin_grp_auth;  
short sql_sysmon_auth;  
short sql_sysmon_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:

- “sqluadav - Get current user’s authorities” on page 487

sql_dir_entry

This structure is used by the DCS directory APIs.

Table 29. Fields in the SQL-DIR-ENTRY Structure

Field Name	Data Type	Description
STRUCT_ID RELEASE CODEPAGE COMMENT LDB TDB AR PARM	SMALLINT SMALLINT CHAR(30) CHAR(8) CHAR(18) CHAR(32) CHAR(512)	Structure identifier. Set to SQL_DCS_STR_ID (defined in sqlenv). Release version (assigned by the API). Code page for comment. Optional description of the database. Local name of the database; must match database alias in system database directory. Actual name of the database. Name of the application client. 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.

API and data structure syntax:

```
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).
```

sql_dir_entry

Related reference:

- “sqlgdad - Catalog a database in the database connection services (DCS) directory” on page 436
- “sqlgdel - Uncatalog a database from the database connection services (DCS) directory” on page 439
- “sqlgde - Get a specific entry in the database connection services (DCS) directory” on page 441
- “sqlgdgt - Get database connection services (DCS) directory entries” on page 443

SQLB_TBS_STATS

This structure is used to return additional table space statistics to an application program.

Table 30. 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.
FREEPAGES	INTEGER	For DMS, equal to USEABLEPAGES minus USED PAGES. 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: This is not 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.

API and data structure syntax:

```
SQL_STRUCTURE SQLB_TBS_STATS
{
    sqluint32 totalPages;
    sqluint32 useablePages;
```

SQLB_TBS_STATS

```
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.  
*
```

Related reference:

- “sqlbgtss - Get table space usage statistics” on page 367

SQLB_TBSCONTQRY_DATA

This structure is used to return container data to an application program.

Table 31. 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.
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).

API and data structure syntax:

```
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;
};
```

SQLB_TBSCONTQRY_DATA

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.
*
```

Related reference:

- “sqlbftcq - Fetch the query data for rows in a table space container” on page 363
- “sqlbstsc - Set table space containers” on page 377
- “sqlbtcq - Get the query data for all table space containers” on page 379

SQLB_TBSPQRY_DATA

This structure is used to return table space data to an application program.

Table 32. 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).
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.

SQLB_TBSPQRY_DATA

Table 32. Fields in the SQLB-TBSPQRY-DATA Structure (continued)

Field Name	Data Type	Description
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.
QUIESCER	Array of SQLB_QUIESCER_DATA structures	Each array entry consists of the quiesce data for a quiesced object.
FSCACHING	UNSIGNED CHAR	File system caching policy to support Direct I/O. This is a 31-bit field.
RESERVED	CHAR(31)	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

All types of permanent data. Regular table space.

SQLB_TBS_LONG

All types of permanent data. Large table space.

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.

API and data structure syntax:

```

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;
  
```

SQLB_TBSPQRY_DATA

```
    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];
    unsigned char fsCaching;
    char reserved[31];
};

SQL_STRUCTURE SQLB QUIESCER_DATA
{
    sqluint32 quiesceId;
    sqluint32 quiesceObject;
};
```

SQLB QUIESCER_DATA data structure parameters:

pad Reserved. Used for structure alignment and should not be populated by user data.

pad1 Reserved. Used for structure alignment and should not be populated by user data.

quiesceId

Input. ID of the tablespace that the quiesced object was created in.

quiesceObject

Input. Object ID of the quiesced object.

COBOL Structure:

```
* File: sqlutbsp.cb1
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-FSCACHING      PIC X(1).  
    05 SQL-RESERVED      PIC X(31).
```

*

Related reference:

- “sqlbftpq - Fetch the query data for rows in a table space” on page 365
- “sqlbstpq - Get information about a single table space” on page 375
- “sqlefmem - Free the memory allocated by the sqlbtcq and sqlbmtsq API” on page 432

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 *SQL Reference, Volume 1*

sqlchar

This structure is used to pass variable length data to the database manager.

Table 33. 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.

API and data structure syntax:

```
SQL_STRUCTURE sqlchar
{
    short length;
    _SQLOLDCHAR data[1];
};
```

COBOL Structure:

This is not defined in any header file. The following is an example that shows how to define the structure in COBOL:

```
* Replace maxlen with the appropriate value:
01 SQLCHAR.
49 SQLCHAR-LEN PIC S9(4) COMP-5.
49 SQLCHAR-DATA PIC X(maxlen).
```

Related reference:

- “db2Export - Export data from a database” on page 81
- “db2Import - Import data into a table, hierarchy, nickname or view” on page 139
- “db2Load - Load data into a table” on page 205

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 *SQL Reference, Volume 1*

sqldcol

This structure is used to pass variable column information to the db2Export, db2Import, and db2Load APIs.

Table 34. 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.

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 sqlloctpair 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 POSITIONSFIL. 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.

Table 35. 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 36. Fields in the SQLLOCTAB Structure

Field Name	Data Type	Description
LOCPAIR	Array	An array of sqllocpair structures.

Table 37. 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.

API and data structure syntax:

```
SQL_STRUCTURE sqldcol
{
    short dcolmeth;
    short dcolnum;
    struct sqldcoln dcolname[1];
};

SQL_STRUCTURE sqldcoln
{
    short dcolnlen;
    char *dcolnptr;
};
```

sqldcol

```
SQL_STRUCTURE sqlloctab
{
    struct sqllocpair locpair[1];
};

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:

- “db2Import - Import data into a table, hierarchy, nickname or view” on page 139
- “db2Load - Load data into a table” on page 205
- “db2Export - Export data from a database” on page 81

sqle_addn_options

This structure is used to pass information to the sqleaddn API.

Table 38. Fields in the SQLE-ADDN-OPTIONS 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. Note: This option is ignored for system temporary table spaces that are defined to use automatic storage (that is system temporary table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement or where no MANAGED BY CLAUSE was specified at all). For these table spaces, there is no way to defer container creation or choose to create a set of containers like they are defined on another partition. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database.
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.

API and data structure syntax:

```
SQL_STRUCTURE sqle_addn_options
{
    char sqladdid[8];
    sqluint32 tblspace_type;
    SQL_PDB_NODE_TYPE tblspace_node;
};
```

sqle_addn_options

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 a database partition server to the partitioned database environment” on page 388

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 current package path being set or queried. A maximum of 255 characters can be set, although servers can truncate this to some platform-specific value.
- Client program ID being set or queried. A maximum of 80 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 39. 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.

sqle_client_info

Table 39. Fields in the SQLE-CLIENT-INFO Structure (continued)

Field Name	Data Type	Description
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 40. 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 z/OS 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 z/OS 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 z/OS servers support up to length 32.
SQLE_CLIENT_INFO_PROGRAMID	CHAR(80)	The program identifier for the client. Once this element is set, DB2 UDB for z/OS Version 8 associates this identifier with any statements inserted into the dynamic SQL statement cache. This element is only supported for applications accessing DB2 UDB for z/OS Version 8.
SQLE_CLIENT_INFO_ACCTSTR	CHAR(200)	The accounting string for the client. Some servers may truncate the value. For example, DB2 for z/OS servers support up to length 200.
SQLE_CLIENT_INFO_AUTOCOMMIT	CHAR(1)	The autocommit setting of the client. It can be set to SQLE_CLIENT_INFO_AUTOCOMMIT_ON or SQLE_CLIENT_INFO_AUTOCOMMIT_OFF.

Note: These field names are defined for the C programming language. There are similar names for FORTRAN and COBOL, which have the same semantics.

API and data structure syntax:

```
SQL_STRUCTURE sqle_client_info
{
    unsigned short    type;
    unsigned short    length;
    char *pValue;
};
```

COBOL Structure:

```
* File: sqlenv.cbl
01 SQLC-CLIENT-INFO.
   05 SQLC-CLIENT-INFO-ITEM OCCURS 4 TIMES.
       10 SQLC-CLIENT-INFO-TYPE PIC S9(4) COMP-5.
       10 SQLC-CLIENT-INFO-LENGTH PIC S9(4) COMP-5.
       10 SQLC-CLIENT-INFO-VALUE USAGE IS POINTER.
*
```

Related reference:

- “sqlcset - Set accounting string” on page 466
- “sqlcseti - Set client information” on page 473
- “sqlcqryi - Query client information” on page 464

sqle_conn_setting

This structure is used to specify connection setting types and values for the sqleqryc and sqleetc APIs.

Table 41. Fields in the SQLE-CONN-SETTING Structure

Field Name	Data Type	Description
TYPE VALUE	SMALLINT SMALLINT	Setting type. 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 42. Connection Settings

Type	Value	Description
SQL_CONNECT_TYPE	SQL_CONNECT_1 SQL_CONNECT_2	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). Type 2 CONNECTs support the multiple databases per unit of work semantics of DUOW.
SQL_RULES	SQL_RULES_DB2 SQL_RULES_STD	Enable the SQL CONNECT statement to switch the current connection to an established (dormant) connection. 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.
SQL_DISCONNECT	SQL_DISCONNECT_EXPL SQL_DISCONNECT_COND SQL_DISCONNECT_AUTO	Removes those connections that have been explicitly marked for release by the SQL RELEASE statement at commit. 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. Breaks all connections at commit.

Table 42. Connection Settings (continued)

Type	Value	Description
SQL_SYNCPOINT	SQL_SYNC_TWOPHASE SQL_SYNC_ONEPHASE SQL_SYNC_NONE	Requires a Transaction Manager (TM) to coordinate two-phase commits among databases that support this protocol. Uses one-phase commits to commit the work done by each database in multiple database transactions. Enforces single updater, multiple read behavior. 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 SQL_DEFERRED_PREPARE_YES SQL_DEFERRED_PREPARE_ALL	The PREPARE statement will be executed at the time it is issued. 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. 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.

Table 42. Connection Settings (continued)

Type	Value	Description
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.
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.

API and data structure syntax:

```
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 connection settings” on page 470
- “sqleqryc - Query client connection settings” on page 462

sqle_node_cplic

This structure is used to catalog CPIC nodes for the sqlectnd API.

Table 43. 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

API and data structure syntax:

```
SQL_STRUCTURE sqle_node_cplic
{
    _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 an entry in the node directory” on page 415

sqle_node_ipxspx

This structure is used to catalog IPX/SPX nodes for the sqlectnd API.

Table 44. 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.

API and data structure syntax:

```
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 an entry in the node directory” on page 415

sqle_node_local

This structure is used to catalog local nodes for the sqlectnd API.

Table 45. 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.

API and data structure syntax:

```
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 an entry in the node directory” on page 415

sqlc_node_netb

This structure is used to catalog NetBIOS nodes for the sqlcctnd API.

Table 46. 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.

Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.

API and data structure syntax:

```
SQL_STRUCTURE sqlc_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:

- “sqlcctnd - Catalog an entry in the node directory” on page 415

sqle_node_npipe

This structure is used to catalog named pipe nodes for the sqlectnd API.

Table 47. 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.

API and data structure syntax:

```
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 an entry in the node directory” on page 415

sqle_node_struct

This structure is used to catalog nodes for the sqlectnd API.

Table 48. 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

API and data structure syntax:

```
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 an entry in the node directory” on page 415

sqlc_node_tcpip

This structure is used to catalog TCP/IP nodes for the sqlcnd API.

Note: To catalog a TCP/IP, TCP/IPv4 or TCP/IPv6 node, set the PROTOCOL type in the node directory structure to SQL_PROTOCOL_TCP, SQL_PROTOCOL_TCP4 or SQL_PROTOCOL_TCP6 respectively in the SQLE-NODE-STRUCT structure before calling the sqlcnd API. To catalog a TCP/IP or TCP/IPv4 SOCKS node, set the PROTOCOL type in the node directory structure to SQL_PROTOCOL SOCKS or SQL_PROTOCOL SOCKS4 respectively in the SQLE-NODE-STRUCT structure before calling the sqlcnd API. SOCKS is not supported on IPv6. For example, SQL_PROTOCOL SOCKS with an IPv6 address is not supported.

Table 49. Fields in the SQLE-NODE-TCP structure

Field Name	Data Type	Description
HOSTNAME	CHAR(255)	Hostname or IP address on which the DB2 server instance resides. The type of IP address accepted depends on the protocol selected.
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.

API and data structure syntax:

```
SQL_STRUCTURE sqlc_node_tcpip
{
    _SQLOLDCHAR hostname[SQL_HOSTNAME_SZ+1];
    _SQLOLDCHAR service_name[SQL_SERVICE_NAME_SZ+1];
};
```

COBOL Structure:

```
* File: sqlenv.cbl
01 SQL-NODE-TCP.
   05 HOSTNAME          PIC X(255).
   05 FILLER            PIC X.
   05 SERVICE-NAME     PIC X(14).
   05 FILLER            PIC X.
*
```

Related reference:

- “sqlcnd - Catalog an entry in the node directory” on page 415
- “sqlc_node_struct ” on page 546

sqlebdbesc

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 50. 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: Specify SQL_CS_NONE to specify that the collating sequence for the database is IDENTITY (which implements a binary collating sequence). SQL_CS_NONE is the default.
SQLDBUDC	CHAR(256)	The nth 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.
SQLDBSGP	INTEGER	Reserved field. No longer used.

Table 50. Fields in the SQLEDBDESC Structure (continued)

Field Name	Data Type	Description
SQLDBNSG	SHORT	A value that 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 51. 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.
SQLFSCACHING	UNSIGNED	CHAR File system caching. If a value of 1 is supplied, file system caching will be OFF for the current tablespace. If a value of 0 is supplied, file system caching will be ON for the current tablespace. Any other value will be treated as file system caching ON.
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.
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.

Table 51. Fields in the SQLETSDESC Structure (continued)

Field Name	Data Type	Description
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 SQLETSDESC structures.

Table 52. Fields in the SQLETSDESC 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

For non-Unicode databases, this is the default option, with the collating sequence based on the database territory. For Unicode databases, this option is equivalent to the IDENTITY option.

SQL_CS_USER

Collation sequence is specified by the 256-byte weight table supplied by the user. Each weight in the table is one byte in length.

SQL_CS_NONE

Identity collating sequence, in which strings are compared byte for byte. This is the default for Unicode databases.

SQL_CS_COMPATABILITY

Use pre-Version collating sequence.

SQL_CS_SYSTEM-NLSCHAR

Collating sequence from system using the NLS version of compare routines for character types. This value can only be specified when creating a Thai TIS620-1 database.

SQL_CS_USER-NLSCHAR

Collation sequence is specified by the 256-byte weight table supplied by the user. Each weight in the table is one byte in length. This value can only be specified when creating a Thai TIS620-1 database.

SQL_CS_IDENTITY_16BIT

CESU-8 (Compatibility Encoding Scheme for UTF-16: 8-Bit) collation sequence as specified by the Unicode Technical Report #26, available at the

Unicode Consortium web site (www.unicode.org). This value can only be specified when creating a Unicode database.

SQL_CS_UCA400_NO

UCA (Unicode Collation Algorithm) collation sequence based on the Unicode Standard version 4.00 with normalization implicitly set to 'on'. Details of the UCA can be found in the Unicode Technical Standard #10 available at the Unicode Consortium web site (www.unicode.org). This value can only be specified when creating a Unicode database.

SQL_CS_UCA400_LSK

The UCA (Unicode Collation Algorithm) collation sequence that is based on the Unicode Standard version 4.00 but will sort Slovakian characters in the appropriate order. Details of the UCA can be found in the Unicode Technical Standard #10, which is available at the Unicode Consortium Web site (www.unicode.org). This value can only be specified when creating a Unicode database.

SQL_CS_UCA400_LTH

UCA (Unicode Collation Algorithm) collation sequence based on the Unicode Standard version 4.00, with sorting of all Thai characters according to the Royal Thai Dictionary order. Details of the UCA can be found in the Unicode Technical Standard #10 available at the Unicode Consortium web site (www.unicode.org). This value can only be specified when creating a Unicode database.

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.

API and data structure syntax:

```
SQL_STRUCTURE sqlbdbdesc
{
    _SQLOLDCHAR sqldbdid[8];
    sqlint32 sqldbccp;
    sqlint32 sqldbcss;
    unsigned char sqldbudc[SQL_CS_SZ];
    _SQLOLDCHAR sqldbcmnt[SQL_CMT_SZ+1];
    _SQLOLDCHAR pad[1];
    sqluint32 sqldbsgp;
    short sqldbnsg;
    char pad2[2];
    sqlint32 sqltstxt;
    struct SLETSDESC *sqlcatts;
    struct SLETSDESC *sqlusrts;
    struct SLETSDESC *sqltmpts;
};
```

sqlebdbesc

```
SQL_STRUCTURE SQLETSDESC
{
    char sqltsdid[8];
    sqlint32 sqlextnt;
    sqlint32 sqlprftc;
    double sqlpovhd;
    double sqltrfrt;
    char sqltstyp;
    unsigned char sqlfscaching;
    short sqlccnt;
    struct SQLETSCDESC containr[1];
};

SQL_STRUCTURE SQLETSCDESC
{
    char sqlctype;
    char pad1[3];
    sqlint32 sqlcsize;
    short sqlclen;
    char sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
    char pad2[2];
};
```

sqlebdbesc structure parameters:

pad1 Reserved. Used for structure alignment and should not to be populated by user data.

pad2 Reserved. Used for structure alignment and should not to be populated by user data.

SQLETSCDESC structure parameters:

pad1 Reserved. Used for structure alignment and should not to be populated by user data.

pad2 Reserved. Used for structure alignment and should not to be populated by user data.

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 SOLTSEXT                PIC S9(9) COMP-5.
   05 SQLCATTS                USAGE IS POINTER.
   05 SQLUSRSTS               USAGE IS POINTER.
   05 SQLTMPTS                USAGE IS POINTER.
*
```

```
* File: sqltsd.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.cb1

```

01 SQLETSDESC.
   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 concepts:

- “Unicode implementation in DB2 Database for Linux, UNIX, and Windows” in *Administration Guide: Planning*

Related reference:

- “sqlecrea - Create database” on page 408

sqlebdbdescext

The extended database description block (sqlebdbdescext) structure is used during a call to the sqlecrea API to specify permanent values for database attributes. The extended database description block enables automatic storage for a database, chooses a default page size for the database, or specifies values for new table space attributes that have been introduced. This structure is used in addition to, not instead of, the database description block (sqlebdbdesc) structure.

If this structure is not passed to the sqlecrea API, the following behavior is used:

- Automatic storage is enabled for the database
- The default page size for the database is 4096 bytes (4 KB)
- If relevant, DB2 database systems determine the value of the extended table space attributes automatically

API and data structure syntax:

```
SQL_STRUCTURE sqlebdbdescext
{
    sqluint32 sqlPageSize;
    struct sqlAutoStorageCfg *sqlAutoStorage;
    struct SQLETSDESCEXT *sqlcattsext;
    struct SQLETSDESCEXT *sqlusrtsext;
    struct SQLETSDESCEXT *sqltmptsext;
    void *reserved;
};

SQL_STRUCTURE sqlAutoStorageCfg
{
    char sqlEnableAutoStorage;
    char pad[3];
    sqluint32 sqlNumStoragePaths;
    char **sqlStoragePaths;
};

SQL_STRUCTURE SQLETSDESCEXT
{
    sqlint64 sqlInitSize;
    sqlint64 sqlIncreaseSize;
    sqlint64 sqlMaximumSize;
    char sqlAutoResize;
    char sqlInitSizeUnit;
    char sqlIncreaseSizeUnit;
    char sqlMaximumSizeUnit;
};

SQL_STRUCTURE sqledboptions
{
    void *piAutoConfigInterface;
    sqlint32 restrictive;
    void *reserved;
};
```

sqlebdbdescext data structure parameters:

Table 53. Fields in the sqlebdbdescxt structure

Field name	Data type	Description
SQLPAGESIZE	sqluint32	Specifies the page size of the default buffer pool as well as the initial table spaces (SYSCATSPACE, TEMPSPACE1, USERSPACE1) when the database is created. The value given also represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. See the information that follows this table for values.
SQLAUTOSTORAGE	Pointer	A pointer to an automatic storage configuration structure. This pointer enables or disables automatic storage for the database. If a pointer is given, automatic storage may be enabled or disabled. If NULL, automatic storage is enabled and a single storage path is assumed with a value determined by the dbpath passed in, or the database manager configuration parameter, dftdbpath.
SQLCATTSEXT	Pointer	A pointer to an extended table space description control block (SQLETSDESCEXT) for the system catalog table space, which defines additional attributes to those found in SQLETSDESC. If NULL, the database manager determines the value of these attributes automatically (if relevant).
SQLUSRTSEXT	Pointer	A pointer to an extended table space description control block (SQLETSDESCEXT) for the user table space, which defines additional attributes to those found in SQLETSDESC. If NULL, the database manager determines the value of these attributes automatically (if relevant).

Table 53. Fields in the sqlebdbdesext structure (continued)

Field name	Data type	Description
SQLTMPTSEXT	Pointer	A pointer to an extended table space description control block (SQLETSDESCEXT) for the system temporary table space, which defines additional attributes to those found in SQLETSDESC. If NULL, the database manager determines the value of these attributes automatically (if relevant).
RESERVED	Pointer	A pointer to a database options control block (sqledboptions).

Valid values for SQLPAGESIZE (defined in sqlenv) are:

SQL_PAGESIZE_4K

Default page size for the database is 4 096 bytes.

SQL_PAGESIZE_8K

Default page size for the database is 8 192 bytes.

SQL_PAGESIZE_16K

Default page size for the database is 16 384 bytes.

SQL_PAGESIZE_32K

Default page size for the database is 32 768 bytes.

Automatic storage configuration (sqleAutoStorageCfg) data structure parameters:

The automatic storage configuration (sqleAutoStorageCfg) structure can be used during a call to the sqlecrea API. It is an element of the sqlebdbdesext structure, and it specifies whether or not automatic storage is enabled for the database.

Table 54. Fields in the sqleAutoStorageCfg Structure

Field name	Data type	Description
SQLENABLEAUTOSTORAGE	CHAR(1)	Specifies whether or not automatic storage is enabled for the database. See the information that follows this table for values.
SQLNUMSTORAGEPATHS	sqluint32	A value indicating the number of storage paths being pointed to by the SQLSTORAGEPATHS array. If the value is 0, the SQLSTORAGEPATHS pointer must be NULL. The maximum number of storage paths is 128 (SQL_MAX_STORAGE_PATHS).

Table 54. Fields in the sqlAutoStorageCfg Structure (continued)

Field name	Data type	Description
SQLSTORAGEPATHS	Pointer	An array of string pointers that point to storage paths. The number of pointers in the array is reflected by SQLNUMSTORAGEPATHS. Set SQLSTORAGEPATHS to NULL if there are no storage paths being provided (in which case, SQLNUMSTORAGEPATHS must be set to 0). The maximum length of each path is 175 characters.

Valid values for SQLEnableAutoStorage (defined in sqlenv) are:

SQL_AUTOMATIC_STORAGE_NO

Automatic storage is disabled for the database. When this value is used, SQLNUMSTORAGEPATHS must be set to 0 and SQLSTORAGEPATHS must be set to NULL.

SQL_AUTOMATIC_STORAGE_YES

Automatic storage is enabled for the database. The storage paths used for automatic storage are specified using the SQLSTORAGEPATHS pointer. If this pointer is NULL, then a single storage path is assumed with a value determined by database manager configuration parameter dftdbpath.

SQL_AUTOMATIC_STORAGE_DFT

The database manager determines whether or not automatic storage is enabled. Currently, the choice is made based on the SQLSTORAGEPATHS pointer. If this pointer is NULL, automatic storage is not enabled, otherwise it is enabled. The default value is equivalent to SQL_AUTOMATIC_STORAGE_YES.

Extended table space description block (SQLETSDESCEXT) structure parameters:

The extended table space description block (SQLETSDESCEXT) structure is used to specify the attributes for the three initial table spaces. This structure is used in addition to, not instead of, the Table Space Description Block (SQLETSDESC) structure.

Table 55. Fields in the SQLETSDESCEXT Structure

Field name	Data type	Description
SQLINITSIZE	sqlint64	Defines the initial size of each table space that uses automatic storage. This field is only relevant for regular or large automatic storage table spaces. Use a value of SQL_TBS_AUTOMATIC_INITSIZE for other table space types or if the intent is to have DB2 automatically determine an initial size. Note: The actual value used by the database manager may be slightly smaller or larger than what was specified. This action is taken to keep sizes consistent across containers in the table space and the value provided may not allow for that consistency.

Table 55. Fields in the SQLETSDESCEXT Structure (continued)

Field name	Data type	Description
SQLINCREASESIZE	sqlint64	Defines the size that the database manager automatically increases the table space by when the table space becomes full. This field is only relevant for table spaces that have auto-resize enabled. Use a value of SQL_TBS_AUTOMATIC_INCSIZE if auto-resize is disabled or if the intent is to have the database manager determine the size increase automatically. Note: The actual value used by the database manager may be slightly smaller or larger than what was specified. This action is taken to keep sizes consistent across containers in the table space and the value provided may not allow for that consistency.
SQLMAXIMUMSIZE	sqlint64	Defines the maximum size to which the database manager automatically increases the table space. Alternately, a value of SQL_TBS_NO_MAXSIZE can be used to specify that the maximum size is "unlimited", in which case the table space can grow to the architectural limit for the table space or until a "filesystem full" condition is encountered. This field is only relevant for table spaces that have auto-resize enabled. Use a value of SQL_TBS_AUTOMATIC_MAXSIZE if auto-resize is disabled or if the intent is to have the database manager determine the maximum size automatically. Note: The actual value used by the database manager may be slightly smaller or larger than what was specified. This action is taken to keep sizes consistent across containers in the table space and the value provided may not allow for that consistency.
SQLAUTORESIZE	CHAR(1)	Specifies whether auto-resize is enabled for the table space or not. See the information that follows this table for values.
SQLINITSIZEUNIT	CHAR(1)	If relevant, indicates whether SQLINITSIZE is being provided in bytes, kilobytes, megabytes, or gigabytes. See the information that follows this table for values.
SQLINCREASESIZEUNIT	CHAR(1)	If relevant, indicates whether SQLINCREASESIZE is being provided in bytes, kilobytes, megabytes, gigabytes, or as a percentage. See the information that follows this table for values.
SQLMAXIMUMSIZEUNIT	CHAR(1)	If relevant, indicates whether SQLMAXIMUMSIZE is being provided in bytes, kilobytes, megabytes, or gigabytes. See the information that follows this table for values.

Valid values for SQLAUTORESIZE (defined in sqlenv) are:

SQL_TBS_AUTORESIZE_NO

Auto-resize is disabled for the table space. This value can only be specified for database-managed space (DMS) table spaces or automatic storage table spaces.

SQL_TBS_AUTORESIZE_YES

Auto-resize is enabled for the table space. This value can only be specified for database-managed space (DMS) table spaces or automatic storage table spaces.

SQL_TBS_AUTORESIZE_DFT

The database manager determines whether or not auto-resize is enabled based on the table space type: auto-resize is turned off for database-managed space (DMS) table spaces and on for automatic storage table spaces. Use this value for system-managed space (SMS) table spaces since auto-resize is not applicable for that type of table space.

Valid values for SQLNITSIZEUNIT, SQLINCREASESIZEUNIT and SQLMAXIMUMSIZEUNIT (defined in sqlenv) are:

SQL_TBS_STORAGE_UNIT_BYTES

The value specified in the corresponding size field is in bytes.

SQL_TBS_STORAGE_UNIT_KILOBYTES

The value specified in the corresponding size field is in kilobytes (1 kilobyte = 1 024 bytes).

SQL_TBS_STORAGE_UNIT_MEGABYTES

The value specified in the corresponding size field is in megabytes (1 megabyte = 1 048 576 bytes)

SQL_TBS_STORAGE_UNIT_GIGABYTES

The value specified in the corresponding size field is in gigabytes (1 gigabyte = 1 073 741 824 bytes)

SQL_TBS_STORAGE_UNIT_PERCENT

The value specified in the corresponding size field is a percentage (valid range is 1 to 100). This value is only valid for SQLINCREASESIZEUNIT.

sqldboptions data structure parameters:

piAutoConfigInterface

Input. A pointer to db2AutoConfigInterface structure which contains information that serves as input for the Configuration Advisor

restrictive

The setting of the restrictive field is stored in the RESTRICT_ACCESS database configuration parameter and will affect all future migrations of this database. That is, when a database is migrated to a subsequent release of DB2, the migration utility checks the RESTRICT_ACCESS database configuration parameter setting to determine whether the restrictive set of default actions needs to be applied to any new objects (for example, new system catalog tables) introduced in the new DB2 release.

The valid values (defined in the sqlenv header file, which is located in the include directory) for this parameter are:

**SQL_DB_RESTRICT_ACCESS_NO or
SQL_DB_RESTRICT_ACCESS_DFT**

Indicates that the database is to be created not using the restrictive set of default actions. This setting will result in the following privileges granted to PUBLIC:

- CREATETAB privilege
- BINDADD privilege
- CONNECT privilege
- IMPLSCHEMA privilege
- EXECUTE with GRANT privilege on all procedures in schema SQLJ
- EXECUTE with GRANT privilege on all functions and procedures in schema SYSPROC
- BIND privilege on all packages created in the NULLID schema
- EXECUTE privilege on all packages created in the NULLID schema
- CREATEIN privilege on schema SQLJ
- CREATEIN privilege on schema NULLID
- USE privilege on tablespace USERSPACE1
- SELECT privilege on the SYSIBM catalog tables
- SELECT privilege on the SYSCAT catalog views
- SELECT privilege on the SYSSTAT catalog views
- UPDATE privilege on the SYSSTAT catalog views

SQL_DB_RESTRICT_ACCESS_YES

Indicates that the database is to be created using the restrictive set of default actions. This means that the grant actions listed above under SQL_DB_RESTRICT_ACCESS_NO do not occur.

reserved

Reserved for future use.

Related reference:

- “db2AutoConfig - Access the Configuration Advisor” on page 35
- “sqlecrea - Create database” on page 408

sqledbterritoryinfo

This structure is used to provide code set and territory options to the sqlecrea API. To use XML features, you must set the SQLDBCODESET field of this structure to UTF-8.

Table 56. Fields in the SQLEDBTERRITORYINFO Structure

Field Name	Data Type	Description
SQLDBCODESET	CHAR(9)	Database code set.
SQLDBLOCALE	CHAR(5)	Database territory.

API and data structure syntax:

```
SQL_STRUCTURE sqledbcountryinfo
{
    char sqldbcodeset[SQL_CODESET_LEN + 1];
    char sqldblocale[SQL_LOCALE_LEN + 1];
};
typedef SQL_STRUCTURE sqledbcountryinfo SQLEDBTERRITORYINFO;
```

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 408

sqleninfo

This structure returns information after a call to the sqlengne API.

Table 57. 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 or IPv4 or IPv6 address 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.

Table 57. Fields in the SLENINFO Structure (continued)

Field Name	Data Type	Description
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.
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

API and data structure syntax:

```
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 filesaver[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;
}
```

sqleninfo

```
    _SQLOLDCHAR chgpwd_lu[SQL_RMTLU_SZ];  
    _SQLOLDCHAR transpn[SQL_TPNAME_SZ];  
    _SQLOLDCHAR lanaddr[SQL_LANADDRESS_SZ];  
};
```

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 the next node directory entry” on page 458

sqlfupd

This structure passes information about database configuration files and the database manager configuration file.

Table 58. 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

UInt64

Unsigned 8-byte integer

float 4-byte floating-point decimal

char(n)

String of length n (not including null termination).

Valid entries for the SQLFUPD token element are listed below:

Table 59. Updatable Database Configuration Parameters

Parameter Name	Token	Token Value	Data Type
app_ctl_heap_sz	SQLF_DBTN_APP_CTL_HEAP_SZ	500	UInt16
applheapsz	SQLF_DBTN_APPLHEAPSZ	51	UInt16
appgroup_mem_sz	SQLF_DBTN_APPGROUP_MEM_SZ	800	UInt32
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
autorestart	SQLF_DBTN_AUTO_RESTART	25	UInt16
avg_appls	SQLF_DBTN_AVG_APPLS	47	UInt16
blk_log_dsk_ful	SQLF_DBTN_BLK_LOG_DSK_FUL	804	UInt16
catalogcache_sz	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32
chnpggs_thresh	SQLF_DBTN_CHNGPGS_THRESH	38	UInt16
database_memory	SQLF_DBTN_DATABASE_MEMORY	803	UInt64
dbheap	SQLF_DBTN_DB_HEAP	58	UInt64
dft_degree	SQLF_DBTN_DFT_DEGREE	301	Sint32
dft_extent_sz	SQLF_DBTN_DFT_EXTENT_SZ	54	UInt32
dft_loadrec_ses	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16

Table 59. Updatable Database Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
dft_prefetch_sz	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16
dft_queryopt	SQLF_DBTN_DFT_QUERYOPT	57	Sint32
dft_refresh_age	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)
dft_sqlmathwarn	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16
dir_obj_name	SQLF_DBTN_DIR_OBJ_NAME	46	char(255)
discover	SQLF_DBTN_DISCOVER	308	UInt16
dl_expint	SQLF_DBTN_DL_EXPINT	350	Sint32
dl_num_copies	SQLF_DBTN_DL_NUM_COPIES	351	UInt16
dl_time_drop	SQLF_DBTN_DL_TIME_DROP	353	UInt16
dl_token	SQLF_DBTN_DL_TOKEN	602	char(10)
dl_upper	SQLF_DBTN_DL_UPPER	603	Sint16
dl_w_expint	SQLF_DBTN_DL_WT_IEXPINT	354	Sint32
dlchktime	SQLF_DBTN_DLCHKTIME	9	UInt32
dyn_query_mgmt	SQLF_DBTN_DYN_QUERY_MGMT	604	UInt16
groupheap_ratio	SQLF_DBTN_GROUPHEAP_RATIO	801	UInt16
indexrec	SQLF_DBTN_INDEXREC	30	UInt16
indexsort	SQLF_DBTN_INDEXSORT	35	UInt16
locklist	SQLF_DBTN_LOCK_LIST	704	UInt64
locktimeout	SQLF_DBTN_LOCKTIMEOUT	34	Sint16
logbufsz	SQLF_DBTN_LOGBUFSZ	33	UInt16
logfilsiz	SQLF_DBTN_LOGFIL_SIZ	92	UInt32
logprimary	SQLF_DBTN_LOGPRIMARY	16	UInt16
logretain	SQLF_DBTN_LOG_RETAIN	23	UInt16
logsecond	SQLF_DBTN_LOGSECOND	17	UInt16
maxappls	SQLF_DBTN_MAXAPPLS	6	UInt16
maxfilop	SQLF_DBTN_MAXFILOP	3	UInt16
maxlocks	SQLF_DBTN_MAXLOCKS	15	UInt16
maxlog	SQLF_DBTN_MAX_LOG	807	UInt16
mincommit	SQLF_DBTN_MINCOMMIT	32	UInt16
mirrorlogpath	SQLF_DBTN_MIRRORLOGPATH	806	char(242)
newlogpath	SQLF_DBTN_NEWLOGPATH	20	char(242)
num_db_backups	SQLF_DBTN_NUM_DB_BACKUPS	601	UInt16
num_freqvalues	SQLF_DBTN_NUM_FREQVALUES	36	UInt16
num_iocleaners	SQLF_DBTN_NUM_IOCLEANERS	37	UInt16
num_ioservers	SQLF_DBTN_NUM_IOSERVERS	39	UInt16
numlogspan	SQLF_DBTN_NUM_LOG_SPAN	808	UInt16
num_quantiles	SQLF_DBTN_NUM_QUANTILES	48	UInt16
overflowlogpath	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)
pckcachesz	SQLF_DBTN_PCKCACHE_SZ	505	UInt32
rec_his_retentn	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16
seqdetect	SQLF_DBTN_SEQDETECT	41	UInt16
sheapthres_shr	SQLF_DBTN_SHEAPTHRES_SHR	802	UInt32
softmax	SQLF_DBTN_SOFTMAX	5	UInt16
sortheap	SQLF_DBTN_SORT_HEAP	52	UInt32

Table 59. Updatable Database Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
stat_heap_sz	SQLF_DBTN_STAT_HEAP_SZ	45	UInt32
stmtheap	SQLF_DBTN_STMTHEAP	53	UInt16
trackmod	SQLF_DBTN_TRACKMOD	703	UInt16
tsm_mgmtclass	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)
tsm_nodename	SQLF_DBTN_TSM_NODENAME	306	char(64)
tsm_owner	SQLF_DBTN_TSM_OWNER	305	char(64)
tsm_password	SQLF_DBTN_TSM_PASSWORD	501	char(64)
userexit	SQLF_DBTN_USER_EXIT	24	UInt16
util_heap_sz	SQLF_DBTN_UTIL_HEAP_SZ	55	UInt32

Valid values for indexrec (defined in sqlutil.h):

- SQLF_INX_REC_SYSTEM (0)
- SQLF_INX_REC_REFERENCE (1)
- SQLF_INX_REC_RESTART (2)
- SQLF_INX_REC_RESTART_NO_REDO (3)
- SQLF_INX_REC_ACCESS_NO_REDO (4)

Valid values for logretain (defined in sqlutil.h):

- SQLF_LOGRETAIN_NO (0)
- SQLF_LOGRETAIN_RECOVERY (1)
- SQLF_LOGRETAIN_CAPTURE (2)

Table 60. Non-updatable Database Configuration Parameters

Parameter Name	Token	Token Value	Data Type
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	UInt16
codepage	SQLF_DBTN_CODEPAGE	101	UInt16
codeset	SQLF_DBTN_CODESET	120	char(9) (see note 1 below)
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)
country/region	SQLF_DBTN_COUNTRY	100	UInt16
database_consistent	SQLF_DBTN_CONSISTENT	111	UInt16
database_level	SQLF_DBTN_DATABASE_LEVEL	124	UInt16
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	UInt16
loghead	SQLF_DBTN_LOGHEAD	105	char(12)
logpath	SQLF_DBTN_LOGPATH	103	char(242)
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	UInt16
numsegs	SQLF_DBTN_NUMSEGS	122	UInt16
release	SQLF_DBTN_RELEASE	102	UInt16
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	UInt16
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	UInt16
territory	SQLF_DBTN_TERRITORY	121	char(5) (see note 2 below)
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	UInt16

Notes:

1. 1. char(17) on HP-UX and Solaris operating system.
2. 2. char(33) on HP-UX and Solaris operating system.

Valid entries for the SQLFUPD token element are listed below:

Table 61. Updatable Database Manager Configuration Parameters

Parameter Name	Token	Token Value	Data Type
agent_stack_sz	SQLF_KTN_AGENT_STACK_SZ	61	Uint16
agentpri	SQLF_KTN_AGENTPRI	26	Sint16
aslheapsz	SQLF_KTN_ASLHEAPSZ	15	Uint32
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
authentication	SQLF_KTN_AUTHENTICATION	78	Uint16
backbufsz	SQLF_KTN_BACKBUFSZ	18	Uint32
catalog_noauth	SQLF_KTN_CATALOG_NOAUTH	314	Uint16
comm_bandwidth	SQLF_KTN_COMM_BANDWIDTH	307	float
conn_elapse	SQLF_KTN_CONN_ELAPSE	508	Uint16
cpuspeed	SQLF_KTN_CPUSPEED	42	float
datalinks	SQLF_KTN_DATALINKS	603	Sint16
dft_account_str	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)
dft_client_adpt	SQLF_KTN_DFT_CLIENT_ADPT	82	Uint16
dft_client_comm	SQLF_KTN_DFT_CLIENT_COMM	77	char(31)
dft_monswitches	SQLF_KTN_DFT_MONSWITCHES	29	Uint16
dft_mon_bufpool	SQLF_KTN_DFT_MON_BUFPOOL	33	Uint16
dft_mon_lock	SQLF_KTN_DFT_MON_LOCK	34	Uint16
dft_mon_sort	SQLF_KTN_DFT_MON_SORT	35	Uint16
dft_mon_stmt	SQLF_KTN_DFT_MON_STMT	31	Uint16
dft_mon_table	SQLF_KTN_DFT_MON_TABLE	32	Uint16
dft_mon_uow	SQLF_KTN_DFT_MON_UOW	30	Uint16
dftdbpath	SQLF_KTN_DFTDBPATH	27	char(215)
diaglevel	SQLF_KTN_DIAGLEVEL	64	Uint16
diagpath	SQLF_KTN_DIAGPATH	65	char(215)
dir_cache	SQLF_KTN_DIR_CACHE	40	Uint16
discover	SQLF_KTN_DISCOVER	304	Uint16
discover_comm	SQLF_KTN_DISCOVER_COMM	305	char(35)
discover_inst	SQLF_KTN_DISCOVER_INST	308	Uint16
dos_rqrioblk	SQLF_KTN_DOS_RQRIOBLK	72	Uint16
fcm_num_buffers	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32
fcm_num_channels	SQLF_KTN_FCM_NUM_CHANNELS	902	Uint32
fed_noauth	SQLF_KTN_FED_NOAUTH	806	Uint16
federated	SQLF_KTN_FEDERATED	604	Sint16
fileservr	SQLF_KTN_FILESERVER	47	char(48)
health_mon	SQLF_KTN_HEALTH_MON	804	Uint16
indexrec	SQLF_KTN_INDEXREC	20	Uint16
initdari_jvm	SQLF_KTN_INITDARI_JVM	602	Sint16
instance_memory	SQLF_KTN_INSTANCE_MEMORY	803	Uint64
intra_parallel	SQLF_KTN_INTRA_PARALLEL	306	Sint16

Table 61. Updatable Database Manager Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
ipx_socket	SQLF_KTN_IPX_SOCKET	71	char(4)
java_heap_sz	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32
jdk11_path	SQLF_KTN_JDK11_PATH	311	char(255)
keepfenced	SQLF_KTN_KEEPPFENCED	81	UInt16
max_connections	SQLF_DBTN_MAX_CONNECTIONS	802	Sint32
max_connretries	SQLF_KTN_MAX_CONNRETRIES	509	UInt16
max_coordagents	SQLF_KTN_MAX_COORDAGENTS	501	Sint32
max_querydegree	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32
max_time_diff	SQLF_KTN_MAX_TIME_DIFF	510	UInt16
maxagents	SQLF_KTN_MAXAGENTS	12	UInt32
maxcagents	SQLF_KTN_MAXCAGENTS	13	Sint32
maxdari	SQLF_KTN_MAXDARI	80	Sint32
maxtofilop	SQLF_KTN_MAXTOTFILOP	45	UInt16
min_priv_mem	SQLF_KTN_MIN_PRIV_MEM	43	UInt32
mon_heap_sz	SQLF_KTN_MON_HEAP_SZ	79	UInt16
nname	SQLF_KTN_NNAME	7	char(8)
notifylevel	SQLF_KTN_NOTIFYLEVEL	605	Sint16
num_initagents	SQLF_KTN_NUM_INITAGENTS	500	UInt32
num_initdaris	SQLF_KTN_NUM_INITDARIS	601	Sint32
num_poolagents	SQLF_KTN_NUM_POOLAGENTS	502	Sint32
numdb	SQLF_KTN_NUMDB	6	UInt16
objectname	SQLF_KTN_OBJECTNAME	48	char(48)
priv_mem_thresh	SQLF_KTN_PRIV_MEM_THRESH	44	Sint32
query_heap_sz	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32
restbufsz	SQLF_KTN_RESTBUFSZ	19	UInt32
resync_interval	SQLF_KTN_RESYNC_INTERVAL	68	UInt16
rqioblk	SQLF_KTN_RQRIOBLK	1	UInt16
sheapthres	SQLF_KTN_SHEAPTHRES	21	UInt32
spm_log_file_sz	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32
spm_max_resync	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32
spm_name	SQLF_KTN_SPM_NAME	92	char(8)
start_stop_time	SQLF_KTN_START_STOP_TIME	511	UInt16
svcname	SQLF_KTN_SVCENAME	24	char(14)
sysadm_group	SQLF_KTN_SYSADM_GROUP	39	char(16)
sysctrl_group	SQLF_KTN_SYSCTRL_GROUP	63	char(16)
sysmaint_group	SQLF_KTN_SYSMAINT_GROUP	62	char(16)
tm_database	SQLF_KTN_TM_DATABASE	67	char(8)
tp_mon_name	SQLF_KTN_TP_MON_NAME	66	char(19)
trust_allclnts	SQLF_KTN_TRUST_ALLCLNTS	301	UInt16
trust_clntauth	SQLF_KTN_TRUST_CLNTAUTH	302	UInt16
udf_mem_sz	SQLF_KTN_UDF_MEM_SZ	69	UInt16

Valid values for authentication (defined in sqlenv.h):

- SQL_AUTHENTICATION_SERVER (0)

- SQL_AUTHENTICATION_CLIENT (1)
- SQL_AUTHENTICATION_DCS (2)
- SQL_AUTHENTICATION_DCE (3)
- SQL_AUTHENTICATION_SVR_ENCRYPT (4)
- SQL_AUTHENTICATION_DCS_ENCRYPT (5)
- SQL_AUTHENTICATION_DCE_SVR_ENC (6)
- SQL_AUTHENTICATION_KERBEROS (7)
- SQL_AUTHENTICATION_KRB_SVR_ENC (8)
- SQL_AUTHENTICATION_NOT_SPEC (255)

SQLF_KTN_DFT_MONSWITCHES is a Uint16 parameter, the bits of which indicate the default monitor switch settings. This allows for the specification of a number of parameters at once. The individual bits making up this composite parameter are:

- Bit 1 (xxxx xxx1): dft_mon_uow
- Bit 2 (xxxx xx1x): dft_mon_stmt
- Bit 3 (xxxx x1xx): dft_mon_table
- Bit 4 (xxxx 1xxx): dft_mon_buffpool
- Bit 5 (xxx1 xxxx): dft_mon_lock
- Bit 6 (xx1x xxxx): dft_mon_sort

Valid values for discover (defined in sqlutil.h):

- SQLF_DSCVR_KNOWN (1)
- SQLF_DSCVR_SEARCH (2)

Valid values for indexrec (defined in sqlutil.h):

- SQLF_INX_REC_SYSTEM (0)
- SQLF_INX_REC_REFERENCE (1)
- SQLF_INX_REC_RESTART (2)
- SQLF_INX_REC_RESTART_NO_REDO (3)
- SQLF_INX_REC_ACCESS_NO_REDO (4)

Valid values for trust_allclnts (defined in sqlutil.h):

- SQLF_TRUST_ALLCLNTS_NO (0)
- SQLF_TRUST_ALLCLNTS_YES (1)
- SQLF_TRUST_ALLCLNTS_DRDAONLY (2)

Table 62. Non-updatable Database Manager Configuration Parameters

Parameter Name	Token	Token Value	Data Type
nodetype	SQLF_KTN_NODETYPE	100	Uint16
release	SQLF_KTN_RELEASE	101	Uint16

Valid values for nodetype (defined in sqlutil.h):

- SQLF_NT_STANDALONE (0)
- SQLF_NT_SERVER (1)
- SQLF_NT_REQUESTOR (2)
- SQLF_NT_STAND_REQ (3)

- SQLF_NT_MPP (4)
- SQLF_NT_SATELLITE (5)

API and data structure syntax:

```
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.
*
```

Related concepts:

- “Configuration parameters” in *Performance Guide*

sqllob

This structure is used to represent a LOB data type in a host programming language.

Table 63. Fields in the sqllob structure

Field name	Data type	Description
length	sqluint32	Length in bytes of the data parameter.
data	char(1)	Data being passed in.

API and data structure syntax:

```
SQL_STRUCTURE sqllob
{
    sqluint32 length;
    char data[1];
};
```

Related reference:

- “db2Export - Export data from a database” on page 81

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 64. 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 65. 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). To retrieve a health snapshot with full collection information, specify SQLM_HMON_OPT_COLL_FULL in this field.
OBJ_TYPE	INTEGER	The type of object to be monitored.
OBJECT	CHAR(128)	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 the sqlmon header file, found in the include directory) are:

SQLMA_DB2

Instance related information.

SQLMA_DBASE

Database related information for a particular database. If you use the SQLMA_DBASE value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_APPL

Application information for an application that matches the provided application ID. If you use the SQLMA_APPL value, you must provide an application ID in the object parameter of sqlm_obj_struct structure.

SQLMA_AGENT_ID

Application information for an application that matches the provided agent ID. If you use the SQLMA_AGENT_ID value, you must provide an agent ID in the agent_id parameter of sqlm_obj_struct structure.

SQLMA_DBASE_TABLES

Table information for a particular database. If you use the SQLMA_DBASE_TABLES value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DBASE_APPLS

Application information for all applications connected to a particular database. If you use the SQLMA_DBASE_APPLS value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DBASE_APPLINFO

Summary application information for connections to a particular database. If you use the SQLMA_DBASE_APPLINFO value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DBASE_LOCKS

List of locks held on a particular database. If you use the SQLMA_DBASE_LOCKS value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_APPL_LOCKS

List of locks held by an application with the matching application ID. If you use the SQLMA_APPL_LOCKS value, you must provide an application ID in the object parameter of sqlm_obj_struct structure.

SQLMA_APPL_LOCKS_AGENT_ID

List of locks held by an application with the matching agent ID. If you use the SQLMA_APPL_LOCKS_AGENT_ID value, you must provide an agent ID in the agent_id parameter of sqlm_obj_struct structure.

SQLMA_DBASE_ALL

Database information for all active databases in the instance.

SQLMA_APPL_ALL

Application information for all database connections in the instance.

SQLMA_APPLINFO_ALL

Summary application information for all connections to the instance.

SQLMA_DCS_APPLINFO_ALL

List of Database Connection Services (DCS) connections to the instance.

SQLMA_DYNAMIC_SQL

Dynamic SQL statement information for a particular database. If you use the SQLMA_DYNAMIC_SQL value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DCS_DBASE

Information for a particular Database Connection Services (DCS) database. If you use the SQLMA_DCS_DBASE value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DCS_DBASE_ALL

Information for all active Database Connection Services (DCS) databases.

SQLMA_DCS_APPL_ALL

Database Connection Services (DCS) application information for all connections.

SQLMA_DCS_APPL

Database Connection Services (DCS) application information for an application that matches the provided application ID. If you use the SQLMA_DCS_APPL value, you must provide an application ID in the object parameter of sqlm_obj_struct structure.

SQLMA_DCS_APPL_HANDLE

Database Connection Services (DCS) application information for an application that matches the provided agent ID. If you use the SQLMA_DCS_APPL_HANDLE value, you must provide an agent ID in the agent_id parameter of sqlm_obj_struct structure.

SQLMA_DCS_DBASE_APPLS

Database Connection Services (DCS) application information for all active

connections to a particular database. If you use the SQLMA_DCS_DBASE_APPLS value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DBASE_TABLESPACES

Table space information for a particular database. If you use the SQLMA_DBASE_TABLESPACES value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DBASE_BUFFERPOOLS

Bufferpool information for a particular database. If you use the SQLMA_DBASE_BUFFERPOOLS value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_BUFFERPOOLS_ALL

Information for all bufferpools.

SQLMA_DBASE_REMOTE

Remote access information for a particular federated database. If you use the SQLMA_DBASE_REMOTE value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_DBASE_REMOTE_ALL

Remote access information for all federated databases.

SQLMA_DBASE_APPLS_REMOTE

Remote access information for an application connected to a particular federated database. If you use the SQLMA_DBASE_APPLS_REMOTE value, you must provide the database name in the object parameter of sqlm_obj_struct structure.

SQLMA_APPL_REMOTE_ALL

Remote access information for all applications.

API and data structure syntax:

```
typedef struct sqlma
{
    sqluint32 obj_num;
    sqlm_obj_struct obj_var[1];
}sqlma;

typedef struct sqlm_obj_struct
{
    sqluint32 agent_id;
    sqluint32 obj_type;
    _SQLLOLDCHAR 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(128).
*
```

Related reference:

sqlma

- “db2GetSnapshot - Get a snapshot of the database manager operational status” on page 113
- “db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API” on page 117

sqlopt

This structure is used to pass bind options to the sqlabndx API, precompile options to the sqlaprep API, and rebind options to the sqlarbind API.

Table 66. Fields in the SQLOPT Structure

Field Name	Data Type	Description
HEADER	Structure	An sqloptheader structure.
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 67. 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 68. Fields in the SQLOPTIONS Structure

Field Name	Data Type	Description
TYPE VAL	INTEGER INTEGER	Bind/precompile/rebind option type. Bind/precompile/rebind option value.

Note: The TYPE and VAL fields are repeated for each bind, precompile, or rebind option specified.

API and data structure syntax:

```
SQL_STRUCTURE sqlopt
{
    SQL_STRUCTURE sqloptheader header;
    SQL_STRUCTURE sqloptions option[1];
};

SQL_STRUCTURE sqloptheader
{
    sqluint32 allocated;
    sqluint32 used;
};

SQL_STRUCTURE sqloptions
{
    sqluint32 type;
    sqluintptr val;
};
```

sqlopt

COBOL Structure:

```
* File: sql.cb1
01 SQLOPT.
   05 SQLOPTHEADER.
      10 ALLOCATED    PIC 9(9) COMP-5.
      10 USED        PIC 9(9) COMP-5.
   05 SQLOPTOPTIONS 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 application program to create a package” on page 350
- “sqlaprep - Precompile application program” on page 355
- “sqlarbnd - Rebind package” on page 358

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. An LSN represents the byte offset of the log record from the beginning of the database log.

Table 69. 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.

API and data structure syntax:

```
typedef union SQLU_LSN
{
    unsigned char   lsnChar[6];
    unsigned short  lsnWord[3];
} SQLU_LSN;
```

Related reference:

- “db2ReadLog - Extracts log records” on page 254

sqlu_media_list

This structure is used to pass information to the db2Load API.

Table 70. 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.
FILLER	CHAR(3)	Filler used for proper alignment of data structure in memory.

Table 71. 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.
PREMOTEFETCH	Pointer	A pointer to an sqlu_remotefetch_entry structure

Table 72. Fields in the SQLU-MEDIA-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN MEDIA_ENTRY	INTEGER CHAR(215)	Length of the media_entry field. For languages other than C. Path for a backup image used by the backup and restore utilities.

Table 73. 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.

Table 73. Fields in the SQLU-VENDOR Structure (continued)

Field Name	Data Type	Description
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 74. 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 75. 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.

Table 76. Fields in the SQLU-REMOTEFETCH-ENTRY Structure

Field Name	Data Type	Description
pDatabaseName	Pointer	Source Database Name.
iDatabaseNameLen	INTEGER	Source Database Name Length
pUserID	Pointer	Pointer to UserID.
iUserIDLen	INTEGER	UserID Length.
pPassword	Pointer	Pointer to Password.
iPasswordLen	INTEGER	Password Length.
pTableSchema	Pointer	Pointer to schema of source table.
iTableSchemaLen	INTEGER	Schema Length.
pTableName	Pointer	Pointer to name of source table.
iTableNameLen	INTEGER	Source table name Length.
pStatement	Pointer	Pointer to name of statement.
iStatementLen	INTEGER	Statement Length.
pIsolationLevel	Pointer	Ptr to isolation level (default CS).

Valid values for MEDIA_TYPE (defined in sqlutil) are:

sqlu_media_list

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). Can be specified only for the piSourceList parameter or the piLobFileList parameter.

SQLU_SQL_STMT

SQL query (load only). Can be specified only for the piSourceList parameter.

SQLU_TSM_MEDIA

TSM

SQLU_XBSA_MEDIA

XBSA

SQLU_OTHER_MEDIA

Vendor library

SQLU_REMOTEFETCH

Remote Fetch media (load only). Can be specified only for the piSourceList parameter.

SQLU_USER_EXIT

User exit (OS/2 only)

SQLU_DISK_MEDIA

Disk (for vendor APIs only)

SQLU_DISKETTE_MEDIA

Diskette (for vendor APIs only)

SQLU_NULL_MEDIA

Null (generated internally by the DB2 database)

SQLU_TAPE_MEDIA

Tape (for vendor APIs only).

SQLU_PIPE_MEDIA

Named pipe (for vendor APIs only)

API and data structure syntax:

```
typedef SQL_STRUCTURE sqlu_media_list
{
    char media_type;
    char filler[3];
    sqlint32 sessions;
    union sqlu_media_list_targets target;
} sqlu_media_list;

union sqlu_media_list_targets
{
    struct sqlu_media_entry *media;
    struct sqlu_vendor *vendor;
    struct sqlu_location_entry *location;
    struct sqlu_statement_entry *pStatement;
    struct sqlu_remotefetch_entry *pRemoteFetch;
};

typedef SQL_STRUCTURE sqlu_media_entry
```

```

{
    sqluint32 reserve_len;
    char media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;

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;

typedef SQL_STRUCTURE sqlu_location_entry
{
    sqluint32 reserve_len;
    char location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_entry;

typedef SQL_STRUCTURE sqlu_statement_entry
{
    sqluint32 length;
    char *pEntry;
} sqlu_statement_entry;

typedef SQL_STRUCTURE sqlu_remotefetch_entry
{
    char *pDatabaseName;
    sqluint32 iDatabaseNameLen;
    char *pUserID;
    sqluint32 iUserIDLen;
    char *pPassword;
    sqluint32 iPasswordLen;
    char *pTableSchema;
    sqluint32 iTableSchemaLen;
    char *pTableName;
    sqluint32 iTableNameLen;
    char *pStatement;
    sqluint32 iStatementLen;
    sqlint32 *pIsolationLevel;
    sqluint32 *pEnableParallelism;
} sqlu_remotefetch_entry;

```

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.
*

```

sqlu_media_list

```
* 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.
*
```

Related reference:

- “db2Load - Load data into a table” on page 205

SQLU_RLOG_INFO

This structure contains information about the status of calls to the db2ReadLog API; and to the database log.

Table 77. 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.

API and data structure syntax:

```
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 - Extracts log records” on page 254
- “SQLU_LSN ” on page 581

sqlupi

This structure is used to store partitioning information, such as the distribution map and the distribution key of a table.

Table 78. Fields in the SQLUPI Structure

Field Name	Data Type	Description
PMAPLEN	INTEGER	The length of the distribution map in bytes. For a single-node table, the value is <code>sizeof(SQL_PDB_NODE_TYPE)</code> . For a multi-node table, the value is <code>SQL_PDB_MAP_SIZE * sizeof(SQL_PDB_NODE_TYPE)</code> .
PMAP	SQL_PDB_NODE_TYPE	The distribution map.
SQLD	INTEGER	The number of used <code>SQLPARTKEY</code> elements; that is, the number of key parts in a distribution key.
SQLPARTKEY	Structure	The description of a distribution column in a distribution key. The maximum number of distribution columns is <code>SQL_MAX_NUM_PART_KEYS</code> .

The following table 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 79. 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 79. 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

sqlpartkey data structure parameter descriptions

sqltype

Input. Data type of the distribution key.

sqllen Input. Data length of the distribution key.

API and data structure syntax:

```
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];
};

SQL_STRUCTURE sqlpartkey
{
    unsigned short  sqltype;
    unsigned short  sqllen;
};
```

Related reference:

- “sqlugrpn - Get the database partition server number for a row” on page 493
- “sqlugtpi - Get table distribution information” on page 496

SQLXA_XID

This structure is used by the transaction APIs to identify XA transactions. sqlxhfrg, sqlxphcm, sqlxphrl, sqlcspqy and db2XaListIndTrans APIs constitute the transaction APIs group. These APIs are used for the management of indoubt transactions.

Table 80. Fields in the SQLXA-XID Structure

Field Name	Data Type	Description
FORMATID GTRID_LENGTH BQUAL_LENGTH DATA	INTEGER INTEGER INTEGER CHAR[128]	XA format ID. Length of the global transaction ID. Length of the branch identifier. 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.

API and data structure syntax:

```
struct sqlxa_xid_t {
    sqlint32 formatID;
    sqlint32 gtrid_length;
    sqlint32 bqual_length;
    char data[SQLXA_XIDDATASIZE];
};
typedef struct sqlxa_xid_t SQLXA_XID;
```

Related reference:

- “db2XaListIndTrans - List indoubt transactions” on page 345
- “sqlcspqy - List DRDA indoubt transactions” on page 381
- “sqlxhfrg - Forget transaction status” on page 501
- “sqlxphcm - Commit an indoubt transaction” on page 502
- “sqlxphrl - Roll back an indoubt transaction” on page 504

Chapter 3. DB2 database system plug-ins for customizing database management

DB2 database system plug-ins for customizing database management

DB2 database products come with plug-in interfaces that you and third-party vendors can use to customize certain database management functions.

Currently, DB2 database systems have three types of plug-ins:

- Security plug-ins for customizing DB2 database system authentication and group membership lookup behavior
- Backup and restore plug-ins for backing up and restoring data onto devices that are not supported by backup and restore facilities provided by DB2 database systems
- Compression plug-in for compressing and decompressing backup images

The functionalities provided through the above three plug-ins come with DB2 database system products, however if you want to customize or augment DB2 database system behavior then you can write your own plug-in or purchase one from a vendor.

Each plug-in is a dynamically loadable library of APIs and data structures. The prototypes for the APIs and data structures are provided by DB2 database systems and the implementation is provided by the vendor. DB2 database systems provide the implementations for some of the APIs and data structures. For a list of plug-in APIs and data structures that are implemented by DB2 database systems, refer to the individual plug-in topic. The implementation is in the form of a shared library on UNIX based systems and a DLL on Windows based platforms. For the actual location of where DB2 database systems look for a particular plug-in, refer to the individual plug-in topic.

A plug-in API differs from a DB2 API (for example, db2Export, db2Backup) in two ways. First, the implementation for a plug-in API, in most cases, is provided by the vendor. Whereas the implementation for a DB2 API is provided by DB2. Second, a plug-in API is called by DB2 whereas a DB2 API is called by the user from a client application. So if a plug-in API topic lists a parameter as input then it means that DB2 fills in a value for the parameter and if the parameter is listed as output then the vendor's implementation of the API is responsible for filling in a value for the parameter.

Related concepts:

- "Security plug-ins" on page 592

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 671
- "DB2 APIs for using compression with backup and restore operations" on page 705

Security plug-ins

Security plug-ins

Security plug-ins	592	Security plug-in problem determination	604
Security plug-in library locations	597	Deploying a group retrieval plug-in	606
Security plug-in naming conventions	598	Deploying a user ID/password plug-in	607
Security plug-in support for two-part user IDs	600	Deploying a GSS-API plug-in	609
32-bit and 64-bit considerations for security plug-ins	603	Deploying a Kerberos plug-in	611
		Restrictions on security plug-ins	613

Security plug-ins

Authentication in DB2 is done using *security plug-ins*. A security plug-in is a dynamically-loadable library that provides authentication security services. DB2 provides the following types of plug-ins:

- Group retrieval plug-in: retrieves group membership information for a given user.
- Client authentication plug-in: manages authentication on a DB2 client.
- Server authentication plug-in: manages authentication on a DB2 server.

DB2 supports two mechanisms for plug-in authentication:

User ID/password authentication

This involves authentication using a user ID and password. The following authentication types are implemented using user ID/password authentication plug-ins:

- CLIENT
- SERVER
- SERVER_ENCRYPT
- DATA_ENCRYPT
- DATA_ENCRYPT_CMP

These authentication types determine how and where authentication of a user occurs. The authentication type used depends on the authentication type specified by the *authentication* database manager configuration parameter. If the SRVCON_AUTH parameter is specified it takes precedence over AUTHENTICATION when dealing with connect or attach operations.

GSS-API authentication

GSS-API is formally known as *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Kerberos authentication is also implemented using GSS-API. The following authentication types are implemented using GSS-API authentication plug-ins:

- KERBEROS
- GSSPLUGIN
- KRB_SERVER_ENCRYPT
- GSS_SERVER_ENCRYPT

KRB_SERVER_ENCRYPT and GSS_SERVER_ENCRYPT support both GSS-API authentication and user ID/password authentication; however, GSS-API authentication is the preferred authentication type.

Note: Authentication types determine how and where a user is authenticated. To use a particular authentication type, update the authentication database manager configuration parameter.

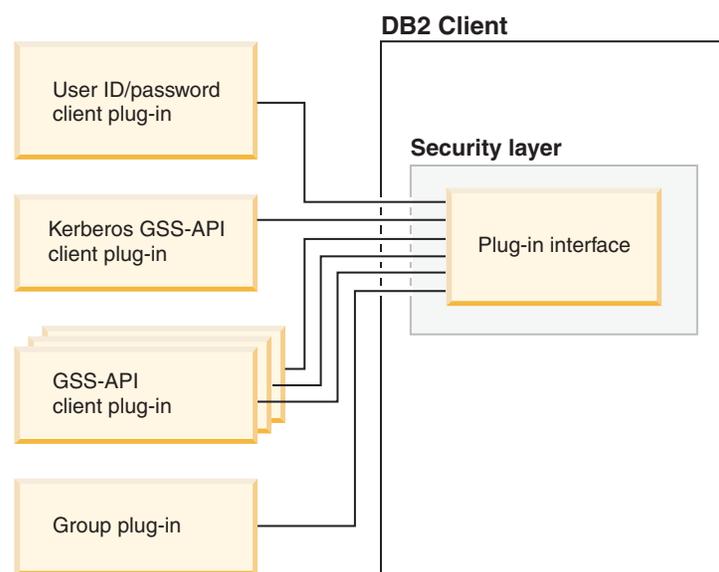
Each of the plug-ins can be used independently or in conjunction with one or more of the other plug-ins. For example, you might only use a server authentication plug-in and assume the DB2 defaults for client and group authentication. Alternatively, you might have only a group or client authentication plug-in. The only situation where both a client and server plug-in are required is for GSS-API authentication plug-ins.

In DB2 Database for Linux, UNIX, and Windows, the default behavior is to use a user ID/password plug-in that implements an operating-system-level mechanism for authentication. In all previous releases of DB2, the default behavior is to directly use operating-system-level authentication without a plug-in implementation. In DB2 Database for Linux, UNIX, and Windows, client-side Kerberos support is available on Solaris, AIX, Windows, and IA32 Linux operating systems; however, it is only enabled by default on Windows.

DB2 database systems include sets of plug-ins for group retrieval, user ID/password authentication, and for Kerberos authentication. With the security plug-in architecture you can customize DB2 client and server authentication behavior by either developing your own plug-ins, or buying plug-ins from a third party.

Deployment of security plug-ins on DB2 clients:

DB2 clients can support one group plug-in, one user ID/password authentication plug-in, and will negotiate with the DB2 server for a particular GSS-API plug-in. This negotiation consists of a scan by the client of the DB2 server's list of implemented GSS-API plug-ins for the first authentication plug-in name that matches an authentication plug-in implemented on the client. The server's list of plug-ins is specified in the *srvcon_gssplugin_list* database manager configuration parameter value, which contains the names of all of the plug-ins that are implemented on the server. The following figure portrays the security plug-in infrastructure on a DB2 client.

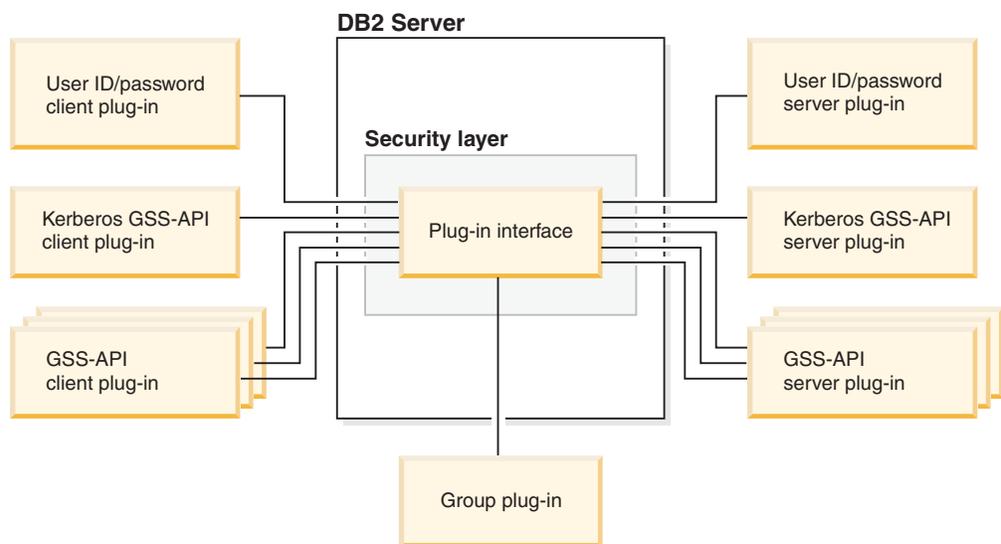


Security plug-ins

Deployment of security plug-ins on DB2 servers:

DB2 servers can support one group plug-in, one user ID/password authentication plug-in, and multiple GSS-API plug-ins. The multiple GSS-API plug-ins are specified in the *srvcon_gssplugin_list* database manager configuration parameter value as a list. Only one GSS-API plug-in in this list can be a Kerberos plug-in.

In addition to server-side security plug-ins, you might also need to deploy client authorization plug-ins on your database server. When you run instance-level operations like **db2start** and **db2trc**, DB2 performs authorization checking for these operations using client authentication plug-ins. Therefore, you should install the client authentication plug-in that corresponds to the server plug-in that is specified by the *authentication* database manager configuration parameter. There is a main distinction between *authentication* and *srvcon_auth*. Specifically, they could be set to different values to cause one mechanism to be used to authenticate database connections and another mechanism to be used for local authorization. The most common usage is *srvcon_auth* set as GSSPLUGIN and *authentication* set as SERVER. If you do not use client authentication plug-ins on the database server, instance level operations such as **db2start** will fail. For example, if the authentication type is SERVER and no user-supplied client plug-in is used, DB2 will use the IBM-shipped default client operating-system plug-in. The following figure portrays the security plug-in infrastructure on a DB2 server.



Note: The integrity of your DB2 database system installation can be compromised if the deployment of security plug-ins are not adequately coded, reviewed, and tested. DB2 database systems take precautions against many common types of failures, but it cannot guarantee complete integrity when user-written security plug-ins are deployed.

Enabling security plug-ins:

The system administrator can specify the names of the plug-ins to use for each authentication mechanism by updating certain plug-in-related database manager configuration parameters. If these parameters are null, they will default to the DB2-supplied plug-ins for group retrieval, user ID/password management, or Kerberos (if authentication is set to Kerberos -- on the server). DB2 does not provide a default GSS-API plug-in. Therefore, if system administrators specify an

authentication type of GSSPLUGIN in *authentication* parameter, they must also specify a GSS-API authentication plug-in in *srocon_gssplugin_list*.

How DB2 loads security plug-ins:

All of the supported plug-ins identified by the database manager configuration parameters are loaded when the database manager starts.

The DB2 client will load a plug-in appropriate for the security mechanism negotiated with the server during connect or attach operations. It is possible that a client application can cause multiple security plug-ins to be concurrently loaded and used. This situation can occur, for example, in a threaded program that has concurrent connections to different databases from different instances.

Actions other than connect or attach operations require authorization (such as updating the database manager configuration, starting and stopping the database manager, turning DB2 trace on and off) as well. For such actions, the DB2 client program will load a plug-in specified in another database manager configuration parameter. If *authentication* is set to GSSPLUGIN, DB2 database manager will use the plug-in specified by *local_gssplugin*. If *authentication* is set to KERBEROS, DB2 database manager will use the plug-in specified by *clnt_krb_plugin*. Otherwise, DB2 database manager will use the plug-in specified by *clnt_pw_plugin*.

Security plug-ins APIs can be called from either an IPv4 platform or an IPv6 platform. An IPv4 address is a 32-bit address which has a readable form a.b.c.d, where each of a-d represents a decimal number from 0-255. An IPv6 address is a 128 bit address of the form a:b:c:d:e:f:g:h, where each of a-h represents 4 hex digits.

Developing security plug-ins:

If you are developing a security plug-in, you need to implement the standard authentication functions that DB2 database manager will use. If you are using your own customized security plug-in, you can use a user ID of up to 255 characters on a connect statement issued through the CLP or a dynamic SQL statement. For the available types of plug-ins, the functionality you will need to implement is as follows:

Group retrieval

Gets the list of groups to which a user belongs.

User ID/password authentication

- Identifies the default security context (client only).
- Validates and optionally changes a password.
- Determines if a given string represents a valid user (server only).
- Modifies the user ID or password provided on the client before it is sent to the server (client only).
- Returns the DB2 authorization ID associated with a given user.

GSS-API authentication

- Implements the required GSS-API functions.
- Identifies the default security context (client only).
- Generates initial credentials based on a user ID and password and optionally changes password (client only).
- Creates and accepts security tickets.

Security plug-ins

- Returns the DB2 authorization ID associated with a given GSS-API security context.

Related concepts:

- “DB2 database system plug-ins for customizing database management” on page 591
- “How DB2 loads security plug-ins” on page 614
- “Security plug-in APIs” on page 625
- “Security plug-in library locations” on page 597
- “Authentication methods for your server” in *Administration Guide: Implementation*

Related reference:

- “Security plug-in samples” on page 671
- “authentication - Authentication type configuration parameter” in *Performance Guide*
- “clnt_krb_plugin - Client Kerberos plug-in configuration parameter” in *Performance Guide*
- “clnt_pw_plugin - Client userid-password plug-in configuration parameter” in *Performance Guide*
- “local_gssplugin - GSS API plug-in used for local instance level authorization configuration parameter” in *Performance Guide*
- “srvcon_auth - Authentication type for incoming connections at the server configuration parameter” in *Performance Guide*
- “srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server configuration parameter” in *Performance Guide*

Security plug-in library locations

After you acquire your security plug-ins (by developing them yourself, or purchasing them from a third party), copy them to specific locations on your database server.

DB2 clients look for client-side user authentication plug-ins in the following directory:

- UNIX 32-bit: \$DB2PATH/security32/plugin/client
- UNIX 64-bit: \$DB2PATH/security64/plugin/client
- WINDOWS 32-bit and 64-bit: \$DB2PATH\security\plugin*instance name*\client

Note: On Windows-based platforms, the subdirectories *instance name* and *client* are not created automatically. The instance owner has to manually create them.

The DB2 database manager looks for server-side user authentication plug-ins in the following directory:

- UNIX 32-bit: \$DB2PATH/security32/plugin/server
- UNIX 64-bit: \$DB2PATH/security64/plugin/server
- WINDOWS 32-bit and 64-bit: \$DB2PATH\security\plugin*instance name*\server

Note: On Windows-based platforms, the subdirectories *instance name* and *server* are not created automatically. The instance owner has to manually create them.

The DB2 database manager looks for group plug-ins in the following directory:

- UNIX 32-bit: \$DB2PATH/security32/plugin/group
- UNIX 64-bit: \$DB2PATH/security64/plugin/group
- WINDOWS 32-bit and 64-bit: \$DB2PATH\security\plugin*instance name*\group

Note: On Windows-based platforms, the subdirectories *instance name* and *group* are not created automatically. The instance owner has to manually create them.

Related concepts:

- “How DB2 loads security plug-ins” on page 614
- “Security plug-in naming conventions” on page 598
- “Security plug-ins” on page 592

Related tasks:

- “Deploying a group retrieval plug-in” on page 606
- “Deploying a GSS-API plug-in” on page 609
- “Deploying a Kerberos plug-in” on page 611
- “Deploying a user ID/password plug-in” on page 607

Security plug-in naming conventions

Security plug-in naming conventions

Security plug-in libraries must have a platform-specific file name extension. Security plug-in libraries written in C or C++ must have a platform-specific file name extension."

- Windows: .dll
- AIX: .a or .so, and if both extensions exist, .a extension is used.
- Linux, HP IPF and Solaris Operating Environment: .so
- HP-UX on PA-RISC: .sl or .so, and if both extensions exist, .sl extension is used.

Note: Users can also develop security plug-ins with the DB2 Universal JDBC Driver.

For example, assume you have a security plug-in library called MyPlugin. For each supported operating system, the appropriate library file name follows:

- Windows 32-bit: MyPlugin.dll
- Windows 64-bit: MyPlugin64.dll
- AIX 32 or 64-bit: MyPlugin.a or MyPlugin.so
- SUN 32 or 64-bit, Linux 32 or 64 bit, HP 32 or 64 bit on IPF: MyPlugin.so
- HP-UX 32 or 64-bit on PA-RISC: MyPlugin.sl or MyPlugin.so

Note: The suffix "64" is only required on the library name for 64-bit Windows security plug-ins.

When you update the database manager configuration with the name of a security plug-in, use the full name of the library without the "64" suffix and omit both the file extension and any qualified path portion of the name. Regardless of the operating system, a security plug-in library called MyPlugin would be registered as follows:

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

The security plug-in name is case sensitive, and must exactly match the library name. DB2 database systems use the value from the relevant database manager configuration parameter to assemble the library path, and then uses the library path to load the security plug-in library.

To avoid security plug-in name conflicts, you should name the plug-in using the authentication method used, and an identifying symbol of the firm that wrote the plug-in. For instance, if the company Foo, Inc. wrote a plug-in implementing the authentication method F00somemethod, the plug-in could have a name like F00somemethod.dll.

The maximum length of a plug-in name (not including the file extension and the "64" suffix) is limited to 32 bytes. There is no maximum number of plug-ins supported by the database server, but the maximum length of the comma-separated list of plug-ins in the database manager configuration is 255 bytes. Two defines located in the include file `sqlenv.h` identifies these two limits:

```
#define SQL_PLUGIN_NAME_SZ 32 /* plug-in name */  
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* GSS API plug-in list */
```

The security plug-in library files must have the following file permissions:

- Owned by the instance owner.
- Readable by all users on the system.

- Executable by all users on the system.

Related concepts:

- “Configuration parameters” in *Performance Guide*
- “Security plug-in library locations” on page 597
- “Security plug-ins” on page 592

Related tasks:

- “Configuring DB2 with configuration parameters” in *Performance Guide*
- “Deploying a group retrieval plug-in” on page 606
- “Deploying a GSS-API plug-in” on page 609
- “Deploying a Kerberos plug-in” on page 611
- “Deploying a user ID/password plug-in” on page 607

Related reference:

- “UPDATE DATABASE MANAGER CONFIGURATION command” in *Command Reference*

Security plug-in support for two-part user IDs

DB2 database for Windows supports the use of two-part user IDs, and the mapping of two-part user IDs to two-part authorization IDs.

For example, consider a Windows operating system two-part user ID composed of a domain and user ID such as: MEDWAY\pieter. In this example, MEDWAY is a domain and pieter is the user name. In DB2 database systems, you can specify whether this two-part user ID should be mapped to either a one-part authorization ID or a two-part authorization ID.

The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior. By default, both one-part user IDs and two-part user IDs map to one-part authorization IDs. The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior. Prior to DB2 UDB, version 8.2, you could only have a one-part user ID that mapped to a one-part authorization ID.

The default mapping of a two-part user ID to a one-part user ID allows a user to connect to the database using:

```
db2 connect to db user MEDWAY\pieter using pw
```

In this situation, if the default behavior is used, the user ID MEDWAY\pieter is resolved to the authorization ID PIETER. If the support for mapping a two-part user ID to a two-part authorization ID is enabled, the authorization ID would be MEDWAY\PIETER.

To enable DB2 to map two-part user IDs to two-part authorization IDs, DB2 supplies two sets of authentication plug-ins:

- One set exclusively maps a one-part user ID to a one-part authorization ID and maps a two-part user-ID to a one-part authorization ID.
- Another set maps both one-part user ID or two-part user ID to a two-part authorization ID.

If a user name in your work environment can be mapped to multiple accounts defined in different locations (such as local account, domain account, and trusted domain accounts), you can specify the plug-ins that enable two-part authorization ID mapping.

It is important to note that a one-part authorization ID, such as, PIETER and a two-part authorization ID that combines a domain and a user ID like MEDWAY\pieter are functionally distinct authorization IDs. The set of privileges associated with one of these authorization IDs can be completely distinct from the set of privileges associated with the other authorization ID. Care should be taken when working with one-part and two-part authorization IDs.

The following table identifies the kinds of plug-ins supplied by DB2 database systems, and the plug-in names for the specific authentication implementations.

Table 81. DB2 security plug-ins

Authentication type	Name of one-part user ID plug-in	Name of two-part user ID plug-in
User ID/password (client)	IBMOSauthclient	IBMOSauthclientTwoPart
User ID/password (server)	IBMOSauthserver	IBMOSauthserverTwoPart

Table 81. DB2 security plug-ins (continued)

Authentication type	Name of one-part user ID plug-in	Name of two-part user ID plug-in
Kerberos	IBMkrb5	IBMkrb5TwoPart

Note: On Windows 64-bit platforms, the characters "64" are appended to the plug-in names listed here.

When you specify an authentication type that requires a user ID/password or Kerberos plug-in, the plug-ins that are listed in the "Name of one-part user ID plug-in" column in the previous table are used by default.

To map a two-part user ID to a two-part authorization ID, you must specify that the two-part plug-in, which is not the default plug-in, be used. Security plug-ins are specified at the instance level by setting the security related database manager configuration parameters as follows:

For server authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBM0SauthserverTwoPart`
- `clnt_pw_plugin` to `IBM0SauthclientTwoPart`

For client authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBM0SauthserverTwoPart`
- `clnt_pw_plugin` to `IBM0SauthclientTwoPart`

For Kerberos authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_gssplugin_list` to `IBM0Skrb5TwoPart`
- `clnt_krb_plugin` to `IBMkrb5TwoPart`

The security plug-in libraries accept two-part user IDs specified in a Microsoft® Windows Security Account Manager compatible format. For example, in the format: `domain\user ID`. Both the domain and user ID information will be used by the DB2 authentication and authorization processes at connection time.

You should consider implementing the two-part plug-ins when creating new databases to avoid conflicts with one-part authorization IDs in existing databases. New databases that use two-part authorization IDs must be created in a separate instance from databases that use single-part authorization IDs.

Related concepts:

- "DB2 and Windows security introduction" in *Administration Guide: Implementation*
- "Security plug-ins" on page 592

Related tasks:

- "Authentication with groups and domain security (Windows)" in *Administration Guide: Implementation*

Related reference:

Security plug-in support for two-part user IDs

- “clnt_krb_plugin - Client Kerberos plug-in configuration parameter” in *Performance Guide*
- “clnt_pw_plugin - Client userid-password plug-in configuration parameter” in *Performance Guide*
- “srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server configuration parameter” in *Performance Guide*
- “srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server configuration parameter” in *Performance Guide*

32-bit and 64-bit considerations for security plug-ins

In general, a 32-bit DB2 instance will use the 32-bit security plug-in and 64-bit DB2 instance will use the 64-bit security plug-in. However, on a 64-bit instance, DB2 supports 32-bit applications, which will require the 32-bit plug-in library.

A database instance where both the 32-bit and the 64-bit applications can run is known as a hybrid instance. If you have a hybrid instance and intend to run 32-bit applications, ensure that the required 32-bit security plug-ins are available in the 32-bit plug-in directory. For 64-bit DB2 instances on Linux and UNIX-based operating systems, excluding Linux on IPF, the directories `security32` and `security64` appear. For a 64-bit DB2 instance on Windows on X64 or IPF, both 32-bit and 64-bit security plug-ins are located in the same directory, but 64-bit plug-in names have a suffix, "64".

If you want to migrate from a 32-bit instance to a 64-bit instance, you should obtain versions of your security plug-ins that are recompiled for 64-bit.

If you acquired your security plug-ins from a vendor that does not supply 64-bit plug-in libraries, you can implement a 64-bit stub that executes a 32-bit application. In this situation, the security plug-in is an external program rather than a library.

Related concepts:

- "Security plug-in library locations" on page 597
- "Security plug-in naming conventions" on page 598
- "Security plug-ins" on page 592

Related tasks:

- "Migrating 32-bit database applications to run on 64-bit instances" in *Migration Guide*

Security plug-in problem determination

Problems with security plug-ins are reported in two ways: through SQL errors and through the administration notification log.

Following are the SQLCODE values related to security plug-ins:

- SQLCODE -1365 is returned when a plug-in error occurs during **db2start** or **db2stop**.
- SQLCODE -1366 is returned whenever there is a local authorization problem.
- SQLCODE -30082 is returned for all connection-related plug-in errors.

The administration notification log is a good resource for debugging and administrating security plug-ins. To see the administration notification log on UNIX, check `sqlllib/db2dump/instance name.nfy`. To see the administration notification log on Windows operating systems, use the Event Viewer tool. The Event Viewer tool can be found by navigating from the Windows operating system "Start" button to Settings -> Control Panel -> Administrative Tools -> Event Viewer. Following are the administration notification log values related to security plug-ins:

- 13000 indicates that a call to a GSS-API security plug-in API failed with an error, and returned an optional error message.
SQLT_ADMIN_GSS_API_ERROR (13000)
Plug-in "*plug-in name*" received error code "*error code*" from GSS API "*gss api name*" with the error message "*error message*"
- 13001 indicates that a call to a DB2 security plug-in API failed with an error, and returned an optional error message.
SQLT_ADMIN_PLUGIN_API_ERROR(13001)
Plug-in "*plug-in name*" received error code "*error code*" from DB2 security plug-in API "*gss api name*" with the error message "*error message*"
- 13002 indicates that DB2 failed to unload a plug-in.
SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
Unable to unload plug-in "*plug-in name*". No further action required.
- 13003 indicates a bad principal name.
SQLT_ADMIN_INVALID_PRIN_NAME (13003)
The principal name "*principal name*" used for "*plug-in name*" is invalid. Fix the principal name.
- 13004 indicates that the plug-in name is not valid. Path separators (On UNIX "/" and on Windows "\\") are not allowed in the plug-in name.
SQLT_ADMIN_INVALID_PLGN_NAME (13004)
The plug-in name "*plug-in name*" is invalid. Fix the plug-in name.
- 13005 indicates that the security plug-in failed to load. Ensure the plug-in is in the correct directory and that the appropriate database manager configuration parameters are updated.
SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
Unable to load plug-in "*plug-in name*". Verify the plug-in existence and directory where it is located is correct.
- 13006 indicates that an unexpected error was encountered by a security plug-in. Gather all the **db2support** information, if possible capture a **db2trc**, and then call IBM support for further assistance.
SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
Plug-in encountered unexpected error. Contact IBM Support for further assistance.

Note: If you are using security plug-ins on a Windows 64-bit database server and are seeing a load error for a security plug-in, see the topics "32-bit and 64-bit considerations for security plug-ins" and "Security plug-in naming conventions". The 64-bit plug-in library requires the suffix "64" on the library name, but the entry in the security plug-in database manager configuration parameters should not indicate this suffix.

Related concepts:

- "32-bit and 64-bit considerations for security plug-ins" on page 603
- "Error information in the SQLCODE, SQLSTATE, and SQLWARN fields" in *Developing Embedded SQL Applications*
- "Security plug-in APIs" on page 625
- "Security plug-in naming conventions" on page 598
- "Security plug-ins" on page 592
- "SQLCODE and SQLSTATE Differences among IBM Relational Database Systems" in *Developing SQL and External Routines*

Related reference:

- "db2trc - Trace command" in *Command Reference*
- "APIs for group retrieval plug-ins" on page 627
- "APIs for user ID/password authentication plug-ins" on page 639
- "Required APIs and definitions for GSS-API authentication plug-ins" on page 667

Deploying a group retrieval plug-in

To customize the DB2 security system's group retrieval behavior, you can develop your own group retrieval plug-in or buy one from a third party.

After you acquire a group retrieval plug-in that is suitable for your database management system, you can deploy it.

Procedure:

To deploy a group retrieval plug-in on the database server, perform the following steps:

1. Copy the group retrieval plug-in library into the server's group plug-in directory.
2. Update the database manager configuration parameter *group_plugin* with the name of the plug-in.

To deploy a group retrieval plug-in on database clients, perform the following steps:

1. Copy the group retrieval plug-in library in the client's group plug-in directory.
2. On the database client, update the database manager configuration parameter *group_plugin* with the name of the plug-in.

Related concepts:

- "Security plug-in library locations" on page 597
- "Security plug-in naming conventions" on page 598
- "Security plug-ins" on page 592

Related tasks:

- "Deploying a GSS-API plug-in" on page 609
- "Deploying a Kerberos plug-in" on page 611
- "Deploying a user ID/password plug-in" on page 607

Related reference:

- "group_plugin - Group plug-in configuration parameter" in *Performance Guide*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Deploying a user ID/password plug-in

To customize the DB2 security system's user ID/password authentication behavior, you can develop your own user ID/password authentication plug-ins or buy one from a third party.

After you acquire user ID/password authentication plug-ins that are suitable for your database management system, you can deploy them.

Depending on their intended usage, all user ID-password based authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used both for local authorization checking and for validating the client when it attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authorization ID exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

In most situations, user ID/password authentication requires only a server-side plug-in. It is possible, though generally deemed less useful, to have only a client user ID/password plug-in. It is possible, though quite unusual to require matching user ID/password plug-ins on both the client and the server.

Procedure:

To deploy a user ID/password authentication plug-in on the database server, perform the following steps on the database server:

1. Copy the user ID/password authentication plug-in library in the server plug-in directory.
2. Update the database manager configuration parameter *srvcon_pw_plugin* with the name of the server plug-in.

This plug-in is used by the server when it is handling CONNECT and ATTACH requests.

3. Either:
 - Set the database manager configuration parameter *srvcon_auth* to the CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP authentication type. Or:
 - Set the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and set *authentication* to CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP authentication type.

To deploy a user ID/password authentication plug-in on database clients, perform the following steps on each client:

1. Copy the user ID/password authentication plug-in library in the client plug-in directory.
2. Update the database manager configuration parameter *clnt_pw_plugin* with the name of the client plug-in.

This plug-in is loaded and called regardless of where the authentication is being done, not only when the database configuration parameter, *authentication* is set to CLIENT.

Deploying a user ID/password plug-in

For local authorization on a client, server, or gateway using a user ID/password authentication plug-in, perform the following steps on each client, server, or gateway:

1. Copy the user ID/password authentication plug-in library in the client plug-in directory on the client, server, or gateway.
2. Update the database manager configuration parameter *clnt_pw_plugin* with the name of the plug-in.
3. Set the *authentication* database manager configuration parameter to CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP.

Related concepts:

- “Security plug-in library locations” on page 597
- “Security plug-in naming conventions” on page 598
- “Security plug-ins” on page 592

Related tasks:

- “Deploying a group retrieval plug-in” on page 606
- “Deploying a GSS-API plug-in” on page 609
- “Deploying a Kerberos plug-in” on page 611

Related reference:

- “authentication - Authentication type configuration parameter” in *Performance Guide*
- “clnt_pw_plugin - Client userid-password plug-in configuration parameter” in *Performance Guide*
- “srvcon_auth - Authentication type for incoming connections at the server configuration parameter” in *Performance Guide*
- “srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server configuration parameter” in *Performance Guide*
- “UPDATE DATABASE MANAGER CONFIGURATION command” in *Command Reference*

Deploying a GSS-API plug-in

To customize the DB2 security system's authentication behavior, you can develop your own authentication plug-ins using the GSS-API, or buy one from a third party.

After you acquire GSS-API authentication plug-ins that are suitable for your database management system, you can deploy them.

In the case of any plug-in types other than Kerberos, you must have matching plug-in names on the client and the server along with the same plug-in type. The plug-ins on the client and server need not be from the same vendor, but they must generate and consume compatible GSS-API tokens. Any combination of Kerberos plug-ins deployed on the client and the server is acceptable since Kerberos plug-ins are standardized. However, different implementations of less standardized GSS-API mechanisms, such as *x.509* certificates, might only be partially compatible with DB2 database systems.

Depending on their intended usage, all GSS-API authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used for local authorization checking and when a client attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authorization ID exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

Procedure:

To deploy a GSS-API authentication plug-in on the database server, perform the following steps on the server:

1. Copy the GSS-API authentication plug-in library in the server plug-in directory. You can copy numerous GSS-API plug-ins into this directory.
2. Update the database manager configuration parameter *srvcn_gssplugin_list* with an ordered, comma-delimited list of the names of the plug-ins installed in the GSS-API plug-in directory.
3. Either:
 - Setting the database manager configuration parameter *srvcn_auth* to GSSPLUGIN or GSS_SERVER_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method. Or:
 - Setting the database manager configuration parameter *srvcn_auth* to NOT_SPECIFIED and setting *authentication* to GSSPLUGIN or GSS_SERVER_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method.

To deploy a GSS-API authentication plug-in on database clients, perform the following steps on each client:

1. Copy the GSS-API authentication plug-in library in the client plug-in directory. You can copy numerous GSS-API plug-ins into this directory. The client selects a GSS-API plug-in for authentication during a CONNECT or ATTACH operation by picking the first GSS-API plug-in contained in the server's plug-in list that is available on the client.

Deploying a GSS-API plug-in

- Optional: Catalog the databases that the client will access, indicating that the client will only accept a GSS-API authentication plug-in as the authentication mechanism. For example:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
```

For local authorization on a client, server, or gateway using a GSS-API authentication plug-in, perform the following steps:

- Copy the GSS-API authentication plug-in library in the client plug-in directory on the client, server, or gateway.
- Update the database manager configuration parameter *local_gssplugin* with the name of the plug-in.
- Set the *authentication* database manager configuration parameter to GSSPLUGIN, or GSS_SERVER_ENCRYPT.

Related concepts:

- “Security plug-in library locations” on page 597
- “Security plug-in naming conventions” on page 598
- “Security plug-ins” on page 592

Related tasks:

- “Deploying a group retrieval plug-in” on page 606
- “Deploying a Kerberos plug-in” on page 611
- “Deploying a user ID/password plug-in” on page 607

Related reference:

- “*srvcon_gssplugin_list* - List of GSS API plug-ins for incoming connections at the server configuration parameter” in *Performance Guide*
- “*authentication* - Authentication type configuration parameter” in *Performance Guide*
- “*local_gssplugin* - GSS API plug-in used for local instance level authorization configuration parameter” in *Performance Guide*
- “*srvcon_auth* - Authentication type for incoming connections at the server configuration parameter” in *Performance Guide*
- “*CATALOG DATABASE* command” in *Command Reference*
- “*UPDATE DATABASE MANAGER CONFIGURATION* command” in *Command Reference*

Deploying a Kerberos plug-in

To customize the DB2 security system's Kerberos authentication behavior, you can develop your own Kerberos authentication plug-ins or buy one from a third party.

After you acquire Kerberos authentication plug-ins that are suitable for your database management system, you can deploy them.

Note that the Kerberos security plug-in will not support IPv6.

Procedure:

To deploy a Kerberos authentication plug-in on the database server, perform the following steps on the server:

1. Copy the Kerberos authentication plug-in library in the server plug-in directory.
2. Update the database manager configuration parameter *srvcn_gssplugin_list*, which is presented as an ordered, comma delimited list, to include the Kerberos server plug-in name. Only one plug-in in this list can be a Kerberos plug-in. If this list is blank and *authentication* is set to KERBEROS or KRB_SVR_ENCRYPT, the default DB2 Kerberos plug-in: IBMkrb5 will be used.
3. Either:
 - Set the database manager configuration parameter *srvcn_auth* to the KERBEROS or KRB_SERVER_ENCRYPT authentication type. (You can deploy a KERBEROS plugin and still use GSSPLUGIN or GSS_SERVER_ENCRYPT) Or:
 - Set the database manager configuration parameter *srvcn_auth* to NOT_SPECIFIED and set *authentication* to KERBEROS or KRB_SERVER_ENCRYPT authentication type.

To deploy a Kerberos authentication plug-in on database clients, perform the following steps on each client:

1. Copy the Kerberos authentication plug-in library in the client plug-in directory.
2. Update the database manager configuration parameter *clnt_krb_plugin* with the name of the Kerberos plug-in.

If *clnt_krb_plugin* is blank, DB2 assumes that the client cannot use Kerberos authentication. This setting is only appropriate when the server cannot support plug-ins. If both the server and the client support security plug-ins, the default server plug-in, *IBMkrb5* would be used over the client value of *clnt_krb_plugin*.

For local authorization on a client, server, or gateway using a Kerberos authentication plug-in, perform the following steps:

- a. Copy the Kerberos authentication plug-in library in the client plug-in directory on the client, server, or gateway.
 - b. Update the database manager configuration parameter *clnt_krb_plugin* with the name of the plug-in.
 - c. Set the *authentication* database manager configuration parameter to KERBEROS, or KRB_SERVER_ENCRYPT.
3. Optional: Catalog the databases that the client will access, indicating that the client will only use a Kerberos authentication plug-in. For example:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
TARGET PRINCIPAL service/host@REALM
```

Deploying a Kerberos plug-in

Note: For platforms supporting Kerberos, the IBMkrb5 library will be present in the client plug-in directory. DB2 will recognize this library as a valid GSS-API plug-in, because Kerberos plug-ins are implemented using GSS-API plug-in.

Related concepts:

- “Security plug-in naming conventions” on page 598
- “Security plug-ins” on page 592
- “Security plug-in library locations” on page 597

Related tasks:

- “Deploying a group retrieval plug-in” on page 606
- “Deploying a GSS-API plug-in” on page 609
- “Deploying a user ID/password plug-in” on page 607

Related reference:

- “authentication - Authentication type configuration parameter” in *Performance Guide*
- “clnt_krb_plugin - Client Kerberos plug-in configuration parameter” in *Performance Guide*
- “srvcon_auth - Authentication type for incoming connections at the server configuration parameter” in *Performance Guide*
- “srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server configuration parameter” in *Performance Guide*
- “CATALOG DATABASE command” in *Command Reference*
- “UPDATE DATABASE MANAGER CONFIGURATION command” in *Command Reference*

Restrictions on security plug-ins

The following are restrictions on the use of security plug-ins:

DB2 database family support restrictions:

You cannot use a GSS-API plug-in to authenticate connections between DB2 clients on Linux, UNIX, and Windows and another DB2 family servers such as DB2 for z/OS. You also cannot authenticate connections from another DB2 database family product, acting as a client, to a DB2 server on Linux, UNIX, or Windows.

If you use a DB2 client on Linux, UNIX, or Windows to connect to other DB2 database family servers, you can use client-side user ID/password plug-ins (such as the IBM-shipped operating system authentication plug-in), or you can write your own user ID/password plug-in. You can also use the built-in Kerberos plug-ins, or implement your own.

With a DB2 client on Linux, UNIX, or Windows, you should not catalog a database using the GSSPLUGIN authentication type.

WebSphere Federation Server support restrictions:

DB2 II does not support the use of delegated credentials from a GSS_API plug-in to establish outbound connections to data sources. Connections to data sources must continue to use the CREATE USER MAPPING command.

Database Administration Server support restrictions:

The DB2 Administration Server (DAS) does not support security plug-ins. The DAS only supports the operating system authentication mechanism.

Security plug-in problem and restriction for DB2 clients (Windows):

When developing security plug-ins that will be deployed in DB2 clients on Windows operating systems, do not unload any auxiliary libraries in the plug-in termination function. This restriction applies to all types of client security plug-ins, including group, user ID and password, Kerberos, and GSS-API plug-ins. Since these termination APIs such as `db2secPluginTerm`, `db2secClientAuthPluginTerm` and `db2secServerAuthPluginTerm` are not called on any Windows platform, you need to do the appropriate resource cleanup.

This restriction is related to cleanup issues associated with the unloading of DLLs on Windows.

Loading plug-in libraries on AIX with extension of .a or .so:

On AIX, security plug-in libraries can have a file name extension of `.a` or `.so`. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of `.a`
Plug-in libraries with file name extensions of `.a` are assumed to be archives containing shared object members. These members must be named `shr.o` (32-bit) or `shr64.o` (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

For example, to build a 32-bit archive style plug-in library:

Restrictions on security plug-ins

```
xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp  
ar rv MyPlugin.a shr.o
```

- Plug-in libraries with a file name extension of .so
Plug-in libraries with file name extensions of .so are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

```
xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

GSS-API security plug-ins do not support message encryption and signing:

Message encryption and signing is not available in GSS-API security plug-ins.

Related reference:

- “db2secClientAuthPluginTerm - Clean up client authentication plug-in resources” on page 648
- “db2secPluginTerm - Clean up group plug-in resources” on page 632
- “db2secServerAuthPluginTerm - Clean up server authentication plug-in resources” on page 664

Developing security plug-ins

How DB2 loads security plug-ins	614	Return codes for security plug-ins	623
Calling sequences for the security plug-in APIs	617	Error message handling for security plug-ins	625
Restrictions for developing security plug-in libraries	620		

This chapter will focus on creating user specific security plug-ins along with restrictions and error messages. To develop a security plug-in, you need an initialization function, and the plug-in must return a integer value to indicate success or failure of the execution of the API.

How DB2 loads security plug-ins

Each plug-in library must contain an initialization function with a specific name determined by the plug-in type:

- Server side authentication plug-in: db2secServerAuthPluginInit()
- Client side authentication plug-in: db2secClientAuthPluginInit()
- Group plug-in: db2secGroupPluginInit()

This function is known as the plug-in initialization function. The plug-in initialization function initializes the specified plug-in and provides DB2 with information that it requires to call the plug-in’s functions. The plug-in initialization function accepts the following parameters:

- The highest version number of the function pointer structure that the DB2 instance invoking the plugin can support
- A pointer to a structure containing pointers to all the APIs requiring implementation
- A pointer to a function that adds log messages to the db2diag.log file
- A pointer to an error message string
- The length of the error message

The following is a function signature for the initialization function of a group retrieval plug-in:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(  
    db2int32 version,  
    void *group_fns,  
    db2secLogMessage *logMessage_fn,  
    char **errmsg,  
    db2int32 *errmsglen);
```

Note: If the plug-in library is compiled as C++, all functions must be declared with: `extern "C"`. DB2 relies on the underlying operating system dynamic loader to handle the C++ constructors and destructors used inside of a C++ user-written plug-in library.

The initialization function is the only function in the plug-in library that uses a prescribed function name. The other plug-in functions are referenced through function pointers returned from the initialization function. Server plug-ins are loaded when the DB2 server starts. Client plug-ins are loaded when required on the client. Immediately after DB2 loads a plug-in library, it will resolve the location of this initialization function and call it. The specific task of this function is as follows:

- Cast the functions pointer to a pointer to an appropriate functions structure
- Fill in the pointers to the other functions in the library
- Fill in the version number of the function pointer structure being returned

DB2 can potentially call the plug-in initialization function more than once. This situation can occur when an application dynamically loads the DB2 client library, unloads it, and reloads it again, then performs authentication functions from a plug-in both before and after reloading. In this situation, the plug-in library might not be unloaded and then re-loaded; however, this behavior varies depending on the operating system.

Another example of DB2 issuing multiple calls to a plug-in initialization function occurs during the execution of stored procedures or federated system calls, where the database server can itself act as a client. If the client and server plug-ins on the database server are in the same file, DB2 could call the plug-in initialization function twice.

If the plug-in detects that `db2secGroupPluginInit` is called more than once, it should handle this event as if it was directed to terminate and reinitialize the plug-in library. As such, the plug-in initialization function should do the entire cleanup tasks that a call to `db2secPluginTerm` would do before returning the set of function pointers again.

On a DB2 server running on a UNIX or Linux-based operating system, DB2 can potentially load and initialize plug-in libraries more than once in different processes.

Related concepts:

- “Security plug-ins” on page 592

Related reference:

- “Calling sequences for the security plug-in APIs” on page 617
- “`db2secClientAuthPluginInit` - Initialize client authentication plug-in” on page 646

How DB2 loads security plug-ins

- “db2secGroupPluginInit - Initialize group plug-in” on page 630
- “db2secPluginTerm - Clean up group plug-in resources” on page 632
- “db2secServerAuthPluginInit - Initialize server authentication plug-in” on page 661
- “Restrictions for developing security plug-in libraries” on page 620

Calling sequences for the security plug-in APIs

There are five main scenarios in which the DB2 database manager will call security plug-in APIs:

- On a client for a database connection (implicit and explicit)
 - CLIENT
 - Server based (SERVER, SERVER_ENCRYPT, DATA_ENCRYPT)
 - GSSAPI and Kerberos
- On a client, server, or gateway for local authorization
- On a server for a database connection
- On a server for a grant statement
- On a server to get a list of groups to which an authorization ID belongs

Note: The DB2 database servers treat database actions requiring local authorizations, such as **db2start**, **db2stop**, and **db2trc** like client applications.

For each of these operations, the sequence with which the DB2 database manager calls the security plug-in APIs is different. Following are the sequences of APIs called by the DB2 database manager for each of these scenarios.

CLIENT - implicit

When the user-configured authentication type is CLIENT, the DB2 client application will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

For an implicit authentication, that is, when you connect without specifying a particular user ID or password, the `db2secValidatePassword` API is called if you are using a user ID/password plug-in. This API permits plug-in developers to prohibit implicit authentication if necessary.

CLIENT - explicit

On an explicit authentication, that is, when you connect to a database in which both the user ID and password are specified, if the *authentication* database manager configuration parameter is set to CLIENT the DB2 client application will call the following security plug-in APIs multiple times if the implementation requires it:

- `db2secRemapUserid();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

Server based (SERVER, SERVER_ENCRYPT, DATA_ENCRYPT) - implicit

On an implicit authentication, when the client and server have negotiated user ID/password authentication (for instance, when the *srvcon_auth* parameter at the server is set to SERVER; SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP), the client application will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secFreeToken();`

Server based (SERVER, SERVER_ENCRYPT, DATA_ENCRYPT) - explicit

On an explicit authentication, when the client and server have negotiated userid/password authentication (for instance, when the *srvcon_auth*

Calling sequences for the security plug-in APIs

parameter at the server is set to SERVER; SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP), the client application will call the following security plug-in APIs:

- `db2secRemapUserid();`

GSSAPI and Kerberos - implicit

On an implicit authentication, when the client and server have negotiated GSS-API or Kerberos authentication (for instance, when the *srvcon_auth* parameter at the server is set to KERBEROS; KRB_SERVER_ENCRYPT, GSSPLUGIN, or GSS_SERVER_ENCRYPT), the client application will call the following security plug-in APIs. (The call to `gss_init_sec_context()` will use `GSS_C_NO_CREDENTIAL` as the input credential.)

- `db2secGetDefaultLoginContext();`
- `db2secProcessServerPrincipalName();`
- `gss_init_sec_context();`
- `gss_release_buffer();`
- `gss_release_name();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

With multi-flow GSS-API support, `gss_init_sec_context()` can be called multiple times if the implementation requires it.

GSSAPI and Kerberos - explicit

If the negotiated authentication type is GSS-API or Kerberos, the client application will call the following security plug-in APIs for GSS-API plug-ins in the following sequence. These APIs are used for both implicit and explicit authentication unless otherwise stated.

- `db2secProcessServerPrincipalName();`
- `db2secGenerateInitialCred();` (For explicit authentication only)
- `gss_init_sec_context();`
- `gss_release_buffer ();`
- `gss_release_name();`
- `gss_release_cred();`
- `db2secFreeInitInfo();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

The API `gss_init_sec_context()` may be called multiple times if a mutual authentication token is returned from the server and the implementation requires it.

On a client, server, or gateway for local authorization

For a local authorization, the DB2 command being used will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

These APIs will be called for both user ID/password and GSS-API authentication mechanisms.

On a server for a database connection

For a database connection on the database server, the DB2 agent process or thread will call the following security plug-in APIs for the user ID/password authentication mechanism:

- `db2secValidatePassword()`; Only if the *authentication* database configuration parameter is not CLIENT
- `db2secGetAuthIDs()`;
- `db2secGetGroupsForUser()`;
- `db2secFreeToken()`;
- `db2secFreeGroupList()`;

For a CONNECT to a database, the DB2 agent process or thread will call the following security plug-in APIs for the GSS-API authentication mechanism:

- `gss_accept_sec_context()`;
- `gss_release_buffer()`;
- `db2secGetAuthIDs()`;
- `db2secGetGroupsForUser()`;
- `gss_delete_sec_context()`;
- `db2secFreeToken()`;
- `db2secFreeGroupList()`;

On a server for a GRANT statement

For a GRANT statement that does not specify the USER or GROUP keyword, (for example, "GRANT CONNECT ON DATABASE TO user1"), the DB2 agent process or thread must be able to determine if user1 is a user, a group, or both. Therefore, the DB2 agent process or thread will call the following security plug-in APIs:

- `db2secDoesGroupExist()`;
- `db2secDoesAuthIDExist()`;

On a server to get a list of groups to which an authid belongs

From your database server, when you need to get a list of groups to which an authorization ID belongs, the DB2 agent process or thread will call the following security plug-in API with only the authorization ID as input:

- `db2secGetGroupsForUser()`;

There will be no token from other security plug-ins.

Related concepts:

- "How DB2 loads security plug-ins" on page 614
- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Restrictions for developing security plug-in libraries

Restrictions for developing security plug-in libraries

Following are restrictions for developing plug-in libraries.

C-linkage

Plug-in libraries must be linked with C-linkage. Header files providing the prototypes, data structures needed to implement the plug-ins, and error code definitions are provided for C/C++ only. Functions that DB2 will resolve at load time must be declared with `extern "C"` if the plug-in library is compiled as C++.

.NET common language runtime is not supported

The .NET common language runtime (CLR) is not supported for compiling and linking source code for plug-in libraries.

Signal handlers

Plug-in libraries must not install signal handlers or change the signal mask, because this will interfere with DB2's signal handlers. Interfering with the DB2 signal handlers could seriously interfere with DB2's ability to report and recover from errors, including traps in the plug-in code itself. Plug-in libraries should also never throw C++ exceptions, as this can also interfere with DB2's error handling.

Thread-safe

Plug-in libraries must be thread-safe and re-entrant. The plug-in initialization function is the only API that is not required to be re-entrant. The plug-in initialization function could potentially be called multiple times from different processes; in which case, the plug-in will cleanup all used resources and reinitialize itself.

Exit handlers and overriding standard C library and operating system calls

Plug-in libraries should not override standard C library or operating system calls. Plug-in libraries should also not install exit handlers or `pthread_atfork` handlers. The use of exit handlers is not recommended because they could be unloaded before the program exits.

Library dependencies

On Linux or UNIX, the processes that load the plug-in libraries can be `setuid` or `setgid`, which means that they will not be able to rely on the `$LD_LIBRARY_PATH`, `$SHLIB_PATH`, or `$LIBPATH` environment variables to find dependent libraries. Therefore, plug-in libraries should not depend on additional libraries, unless any dependant libraries are accessible through other methods, such as the following:

- By being in `/lib` or `/usr/lib`
- By having the directories they reside in being specified OS-wide (such as in the `ld.so.conf` file on Linux)
- By being specified in the `RPATH` in the plug-in library itself

This restriction is not applicable to Windows operating systems.

Symbol collisions

When possible, plug-in libraries should be compiled and linked with any available options that reduce the likelihood of symbol collisions, such as those that reduce unbound external symbolic references. For example, use of the `"-Bsymbolic"` linker option on HP, Sun Solaris, and Linux can help prevent problems related to symbol collisions. However, for plug-ins written on AIX, do not use the `"-brtl"` linker option explicitly or implicitly.

Restrictions for developing security plug-in libraries

32-bit and 64-bit applications

32-bit applications must use 32-bit plug-ins. 64-bit applications must use 64-bit plug-ins. Refer to the topic 32-bit and 64-bit considerations for security plug-ins for more details.

Text strings

Input text strings are not guaranteed to be null-terminated, and output strings are not required to be null-terminated. Instead, integer lengths are given for all input strings, and pointers to integers are given for lengths to be returned.

Passing authorization ID parameters

An authorization ID (authid) parameter that DB2 passes into a plug-in (an input authid parameter) will contain an upper-case authid, with padded blanks removed. An authid parameter that a plug-in returns to DB2 (an output authid parameter) does not require any special treatment, but DB2 will fold the authid to upper-case and pad it with blanks according to the internal DB2 standard.

Size limits for parameters

The plug-in APIs use the following as length limits for parameters:

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERSPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

Note: The maximum AUTHID length accepted by DB2 database systems in v9 is 30 characters.

A particular plug-in implementation may require or enforce smaller maximum lengths for the authorization IDs, user IDs, and passwords. In particular, the operating system authentication plug-ins supplied with DB2 database systems are restricted to the maximum user, group and namespace length limits enforced by the operating system for cases where the operating system limits are lower than those stated above.

Security plug-in library extensions in AIX

On AIX systems, security plug-in libraries can have a file name extension of *.a* or *.so*. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of *.a* are assumed to be archives containing shared object members. These members must be named *shr.o* (32-bit) or *shr64.o* (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

For example, to build a 32-bit archive style plug-in library:

```
xlc_r -qmkshrojb -o shr.o MyPlugin.c -bE:MyPlugin.exp
ar rv MyPlugin.a shr.o
```

- Plug-in libraries with a file name extension of *.so* are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

```
xlc_r -qmkshrojb -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

Related concepts:

Restrictions for developing security plug-in libraries

- “32-bit and 64-bit considerations for security plug-ins” on page 603
- “How DB2 loads security plug-ins” on page 614
- “Security plug-in library locations” on page 597
- “Security plug-in naming conventions” on page 598
- “Security plug-ins” on page 592

Return codes for security plug-ins

All security plug-in APIs must return an integer value to indicate the success or failure of the execution of the API. A return code value of 0 indicates that the API ran successfully. All negative return codes, with the exception of -3, -4, and -5, indicate that the API encountered an error.

All negative return codes returned from the security-plug-in APIs are mapped to SQLCODE -1365, SQLCODE -1366, or SQLCODE -30082, with the exception of return codes with the -3, -4, or -5. The values -3, -4, and -5 are used to indicate whether or not an AUTHORIZATION ID represents a valid user or group.

All the security plug-in API return codes are defined in `db2secPlugin.h`, which can be found in the DB2 include directory: `SQLLIB/include`.

Details regarding all of the security plug-in return codes are presented in the following table:

Table 82. Security plug-in return codes

Return code	Define value	Meaning	Applicable APIs
0	DB2SEC_PLUGIN_OK	The plug-in API executed successfully.	All
-1	DB2SEC_PLUGIN_UNKNOWNEROR	The plug-in API encountered an unexpected error.	All
-2	DB2SEC_PLUGIN_BADUSER	The user ID passed in as input is not defined.	db2secGenerateInitialCred db2secValidatePassword db2secRemapUserid db2secGetGroupsForUser
-3	DB2SEC_PLUGIN_INVALIDUSERORGROUP	No such user or group.	db2secDoesAuthIDExist db2secDoesGroupExist
-4	DB2SEC_PLUGIN_USERSTATUSNOTKNOWN	Unknown user status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group.	db2secDoesAuthIDExist
-5	DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN	Unknown group status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group.	db2secDoesGroupExist
-6	DB2SEC_PLUGIN_UID_EXPIRED	User ID expired.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-7	DB2SEC_PLUGIN_PWD_EXPIRED	Password expired.	db2secValidatePassword db2GetGroupsForUser db2secGenerateInitialCred
-8	DB2SEC_PLUGIN_USER_REVOKED	User revoked.	db2secValidatePassword db2GetGroupsForUser
-9	DB2SEC_PLUGIN_USER_SUSPENDED	User suspended.	db2secValidatePassword db2GetGroupsForUser
-10	DB2SEC_PLUGIN_BADPWD	Bad password.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred
-11	DB2SEC_PLUGIN_BAD_NEWPASSWORD	Bad new password.	db2secValidatePassword db2secRemapUserid
-12	DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED	Change password not supported.	db2secValidatePassword db2secRemapUserid db2secGenerateInitialCred

Return codes for security plug-ins

Table 82. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-13	DB2SEC_PLUGIN_NOMEM	Plug-in attempt to allocate memory failed due to insufficient memory.	All
-14	DB2SEC_PLUGIN_DISKERROR	Plug-in encountered a disk error.	All
-15	DB2SEC_PLUGIN_NOPERM	Plug-in attempt to access a file failed because of wrong permissions on the file.	All
-16	DB2SEC_PLUGIN_NETWORKERROR	Plug-in encountered a network error.	All
-17	DB2SEC_PLUGIN_CANTLOADLIBRARY	Plug-in is unable to load a required library.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-18	DB2SEC_PLUGIN_CANT_OPEN_FILE	Plug-in is unable to open and read a file for a reason other than missing file or inadequate file permissions.	All
-19	DB2SEC_PLUGIN_FILENOTFOUND	Plug-in is unable to open and read a file, because the file is missing from the file system.	All
-20	DB2SEC_PLUGIN_CONNECTION_DISALLOWED	The plug-in is refusing the connection because of the restriction on which database is allowed to connect, or the TCP/IP address cannot connect to a specific database.	All server-side plug-in APIs.
-21	DB2SEC_PLUGIN_NO_CRED	GSS API plug-in only: initial client credential is missing.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	GSS API plug-in only: client credential has expired.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-23	DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME	GSS API plug-in only: the principal name is invalid.	db2secProcessServerPrincipalName
-24	DB2SEC_PLUGIN_NO_CON_DETAILS	This return code is returned by the db2secGetConDetails callback (for example, from DB2 to the plug-in) to indicate that DB2 is unable to determine the client's TCP/IP address.	db2secGetConDetails
-25	DB2SEC_PLUGIN_BAD_INPUT_PARAMETERS	Some parameters are not valid or are missing when plug-in API is called.	All
-26	DB2SEC_PLUGIN_INCOMPATIBLE_VER	The version of the APIs reported by the plug-in is not compatible with DB2.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-27	DB2SEC_PLUGIN_PROCESS_LIMIT	Insufficient resources are available for the plug-in to create a new process.	All
-28	DB2SEC_PLUGIN_NO_LICENSES	The plug-in encountered a user license problem. A possibility exists that the underlying mechanism license has reached the limit.	All

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-in problem determination” on page 604
- “Security plug-ins” on page 592

Error message handling for security plug-ins

When an error occurs in a security plug-in API, the API can return an ASCII text string in the `errmsg` field to provide a more specific description of the problem than the return code. For instance, the `errmsg` string can contain "File /home/db2inst1/mypasswd.txt does not exist." DB2 will write this entire string into the DB2 administration notification log, and will also include a truncated version as a token in some SQL messages. Because tokens in SQL messages can only be of limited length, these messages should be kept short, and important variable portions of these messages should appear at the front of the string. To aid in debugging, consider adding the name of the security plug-in to the error message.

For non-urgent errors, such as password expired errors, the `errmsg` string will only be dumped when the `DIAGLEVEL` database manager configuration parameter is set at 4.

The memory for these error messages must be allocated by the security plug-in. Therefore, the plug-ins must also provide an API to free this memory: `db2secFreeErrorMsg`.

The `errmsg` field will only be checked by DB2 if an API returns a non-zero value. Therefore, the plug-in should not allocate memory for this returned error message if there is no error.

At initialization time a message logging function pointer, `logMessage_fn`, is passed to the group, client, and server plug-ins. The plug-ins can use the function to log any debugging information to `db2diag.log`. For example:

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
                  "db2secGroupPluginInit successful",
                  strlen("db2secGroupPluginInit successful"));
```

For more details about each parameter for the `db2secLogMessage` function, refer to the initialization API for each of the plug-in types.

Related concepts:

- "Security plug-in APIs" on page 625
- "Security plug-in problem determination" on page 604
- "Security plug-ins" on page 592
- "CLI trace files" in *Troubleshooting Guide*

Related reference:

- "Return codes for security plug-ins" on page 623

Security plug-in APIs

Security plug-in APIs	625	GSS-API plug-in APIs	667
Group plug-in APIs	627	Security plug-in API versioning	670
User authentication plug-in APIs	639	Security plug-in samples	671

Security plug-in APIs

To enable you to customize the DB2 database system authentication and group membership lookup behavior, DB2 provides APIs that you can use to modify existing plug-ins or build new security plug-ins.

Error message handling for security plug-ins

When you develop a security plug-in, you need to implement the standard authentication or group membership lookup functions that DB2 will invoke. For the three available types of plug-ins, the functionality you need to implement is as follows:

Group retrieval

Retrieves group membership information for a given user and determines if a given string represents a valid group name.

User ID/password authentication

Authentication that identifies the default security context (client only), validates and optionally changes a password, determines if a given string represents a valid user (server only), modifies the user ID or password provided on the client before it is sent to the server (client only), returns the DB2 authorization ID associated with a given user.

GSS-API authentication

Authentication that implements the required GSS-API functions, identifies the default security context (client side only), generates initial credentials based on user ID and password, and optionally changes password (client side only), creates and accepts security tickets, and returns the DB2 authorization ID associated with a given GSS-API security context.

The following are the definitions for terminology used in the descriptions of the plug-in APIs.

Plug-in

A dynamically loadable library that DB2 will load to access user-written authentication or group membership lookup functions.

Implicit authentication

A connection to a database without specifying a user ID or a password.

Explicit authentication

A connection to a database in which both the user ID and password are specified.

Authid

An internal ID representing an individual or group to which authorities and privileges within the database are granted. Internally, a DB2 authid is folded to upper-case and is a minimum of 8 characters (blank padded to 8 characters). Currently, DB2 requires authids, user IDs, passwords, group names, namespaces, and domain names that can be represented in 7-bit ASCII. The maximum length of an authid is 30 characters.

Local authorization

Authorization that is local to the server or client that implements it, that checks if a user is authorized to perform an action (other than connecting to the database), such as starting and stopping the database manager, turning DB2 trace on and off, or updating the database manager configuration.

Namespace

A collection or grouping of users within which individual user identifiers must be unique. Common examples include Windows domains and Kerberos Realms. For example, within the Windows domain "usa.company.com" all user names must be unique. For example, "user1@usa.company.com". The same user ID in another domain, as in the case of "user1@canada.company.com", however refers to a different person.

A fully qualified user identifier includes a user ID and namespace pair; for example, "user@domain.name" or "domain\user".

Input Indicates that DB2 will fill in the value for the security plug-in API parameter.

Output Indicates that the security plug-in API will fill in the value for the API parameter.

Related concepts:

- "Security plug-ins" on page 592

Related reference:

- "APIs for group retrieval plug-ins" on page 627
- "APIs for user ID/password authentication plug-ins" on page 639
- "Required APIs and definitions for GSS-API authentication plug-ins" on page 667

Group plug-in APIs

APIs for group retrieval plug-ins: For the group retrieval plug-in library, you will need to implement the following APIs:

- db2secGroupPluginInit

Note: The db2secGroupPluginInit API takes as input a pointer, *logMessage_fn, to an API with the following prototype:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
    db2int32 level,
    void *data,
    db2int32 length
);
```

The db2secLogMessage API allows the plug-in to log messages to db2diag.log for debugging or informational purposes. This API is provided by DB2, so you need not implement it.

- db2secPluginTerm
- db2secGetGroupsForUser
- db2secDoesGroupExist
- db2secFreeGroupListMemory
- db2secFreeErrorMsg
- The only API that must be resolvable externally is db2secGroupPluginInit. This API will take a void * parameter, which should be cast to the type:

```
typedef struct db2secGroupFunctions_1
{
    db2int32 version;
    db2int32 pluginType;
    SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser)
    (
        const char *authid,
        db2int32 authidlen,
        const char *userid,
        db2int32 useridlen,
        const char *usernamespace,
        db2int32 usernamespaceLen,
        db2int32 usernamespaceType,
```

APIs for group retrieval plug-ins

```
const char *dbname,
db2int32   dbnameLen,
const void *token,
db2int32   tokentype,
db2int32   location,
const char *authpluginname,
db2int32   authpluginnameLen,
void       **groupList,
db2int32   *numGroups,
char       **errorMsg,
db2int32   *errorMsgLen
);

SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)
(
const char *groupName,
db2int32   groupNameLen,
char       **errorMsg,
db2int32   *errorMsgLen
);

SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)
(
void       *ptr,
char       **errorMsg,
db2int32   *errorMsgLen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *msgtoFree
);

SQL_API_RC (SQL_API_FN * db2secPluginTerm)
(
char       **errorMsg,
db2int32   *errorMsgLen
);

} db2secGroupFunctions_1;
```

The `db2secGroupPluginInit` API assigns the addresses for the rest of the externally available functions.

Note: The `_1` indicates that this is the structure corresponding to version 1 of the API. Subsequent interface versions will have the extension `_2`, `_3`, and so on.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related tasks:

- “Deploying a group retrieval plug-in” on page 606

Related reference:

- “`db2secDoesGroupExist` - Check if group exists” on page 637
- “`db2secFreeErrorMsg` - Free error message memory” on page 639
- “`db2secFreeGroupListMemory` - Free group list memory” on page 638
- “`db2secGetGroupsForUser` - Get list of groups for user” on page 633
- “`db2secGroupPluginInit` - Initialize group plug-in” on page 630

- “db2secPluginTerm - Clean up group plug-in resources” on page 632

db2secGroupPluginInit - Initialize group plug-in

db2secGroupPluginInit - Initialize group plug-in: Initialization API, for the group-retrieval plug-in, that DB2 calls immediately after loading the plug-in.

API and data structure syntax:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit
(
    db2int32 version,
    void *group_fns,
    db2secLogMessage *logMessage_fn,
    char **errmsg,
    db2int32 *errmsglen );
```

db2secGroupPluginInit API parameters:

version

Input. The highest version of the API supported by the instance loading that plugin. The value DB2SEC_API_VERSION (in db2secPlugin.h) contains the latest version number of the API that DB2 currently supports.

group_fns

Output. A pointer to the db2secGroupFunctions_<version_number> (also known as group_functions_<version_number>) structure. The db2secGroupFunctions_<version_number> structure contains pointers to the APIs implemented for the group-retrieval plug-in. In future, there might be different versions of the APIs (for example, db2secGroupFunctions_<version_number>), so the group_fns parameter is cast as a pointer to the db2secGroupFunctions_<version_number> structure corresponding to the version the plug-in has implemented. The first parameter of the group_functions_<version_number> structure tells DB2 the version of the APIs that the plug-in has implemented. Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented. The version number represents the version of the APIs implemented by the plugin, and the pluginType should be set to DB2SEC_PLUGIN_TYPE_GROUP.

logMessage_fn

Input. A pointer to the db2secLogMessage API, which is implemented by DB2. The db2secGroupPluginInit API can call the db2secLogMessage API to log messages to db2diag.log for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag.log file and the last two parameters respectively are the message string and its length. The valid values for the first parameter of db2secLogMessage API (defined in db2secPlugin.h) are:

- DB2SEC_LOG_NONE: (0) No logging
- DB2SEC_LOG_CRITICAL: (1) Severe Error encountered
- DB2SEC_LOG_ERROR: (2) Error encountered
- DB2SEC_LOG_WARNING: (3) Warning
- DB2SEC_LOG_INFO: (4) Informational

The message text will show up in the diag.log only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter. So for example, if you use the DB2SEC_LOG_INFO value, the message text will only show up in the db2diag.log if the diaglevel database manager configuration parameter is set to 4.

errmsg

Output. A pointer to the address of an ASCII error message string allocated

db2secGroupPluginInit - Initialize group plug-in

by the plug-in that can be returned in this parameter if the db2secGroupPluginInit API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for group retrieval plug-ins” on page 627
- “Security plug-in samples” on page 671

db2secPluginTerm - Clean up group plug-in resources

db2secPluginTerm - Clean up group plug-in resources: Frees resources used by the group-retrieval plug-in. This API is called by DB2 just before it unloads the group-retrieval plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform. For further information, please refer to the topic "Restrictions on security plug-ins".

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secPluginTerm)
( char          **errorMsg,
  db2int32 *errormsglen );
```

db2secPluginTerm API parameters:

errorMsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secPluginTerm API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errorMsg parameter.

Related concepts:

- "Restrictions on security plug-ins" on page 613
- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for group retrieval plug-ins" on page 627
- "Security plug-in samples" on page 671
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*

db2secGetGroupsForUser - Get list of groups for user

db2secGetGroupsForUser - Get list of groups for user: Returns the list of groups to which a user belongs.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGetGroupsForUser)
( const char *authid,
  db2int32 authidlen,
  const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespace,
  db2int32 usernamespace,
  const char *dbname,
  db2int32 dbname,
  void *token,
  db2int32 tokentype,
  db2int32 location,
  const char *authpluginname,
  db2int32 authpluginname,
  void **grouplist,
  db2int32 *numgroups,
  char **errmsg,
  db2int32 *errormsglen );
```

db2secGetGroupsForUser API parameters:

authid Input. This parameter value is an SQL authid, which means that DB2 converts it to an uppercase character string with no trailing blanks. DB2 will always provide a non-null value for the authid parameter. The API must be able to return a list of groups to which the authid belongs without depending on the other input parameters. It is permissible to return a shortened or empty list if this cannot be determined.

If a user does not exist, the API must return the return code DB2SEC_PLUGIN_BADUSER. DB2 does not treat the case of a user not existing as an error, since it is permissible for an authid to not have any groups associated with it. For example, the db2secGetAuthids API can return an authid that does not exist on the operating system. The authid is not associated with any groups, however, it can still be assigned privileges directly.

If the API cannot return a complete list of groups using only the authid, then there will be some restrictions on certain SQL functions related to group support. For a list of possible problem scenarios, refer to the Usage notes section in this topic.

authidlen

Input. Length in bytes of the authid parameter value. DB2 will always provide a non-zero value for the authidlen parameter.

userid Input. This is the user ID corresponding to the authid. When this API is called on the server in a non-connect scenario, this parameter will not be filled by DB2.

useridlen

Input. Length in bytes of the userid parameter value.

usernamespace

Input. The namespace from which the user ID was obtained. When the user ID is not available, this parameter will not be filled by DB2.

usernamespace

Input. Length in bytes of the usernamespace parameter value.

db2secGetGroupsForUser - Get list of groups for user

usernamespacetype

Input. The type of namespace. Valid values for the usernamespacetype parameter (defined in db2secPlugin.h) are:

- DB2SEC_NAMESPACE_SAM_COMPATIBLE Corresponds to a username style like domain\myname
- DB2SEC_NAMESPACE_USER_PRINCIPAL Corresponds to a username style like myname@domain.ibm.com

Currently DB2 only supports the value DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, the usernamespacetype parameter is set to the value DB2SEC_USER_NAMESPACE_UNDEFINED (defined in db2secPlugin.h).

dbname

Input. Name of the database being connected to. This parameter can be NULL in a non-connect scenario.

dbnamelen

Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL in a non-connect scenario.

token Input. A pointer to data provided by the authentication plug-in. It is not used by DB2. It provides the plug-in writer with the ability to coordinate user and group information. This parameter might not be provided in all cases (for example, in a non-connect scenario), in which case it will be NULL. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss_ctx_id_t).

tokentype

Input. Indicates the type of data provided by the authentication plug-in. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss_ctx_id_t). If the authentication plug-in used is user ID/password based, it will be a generic type. Valid values for the tokentype parameter (defined in db2secPlugin.h) are:

- DB2SEC_GENERIC: Indicates that the token is from a user ID/password based plug-in.
- DB2SEC_GSSAPI_CTX_HANDLE: Indicates that the token is from a GSS-API (including Kerberos) based plug-in.

location

Input. Indicates whether DB2 is calling this API on the client side or server side. Valid values for the location parameter (defined in db2secPlugin.h) are:

- DB2SEC_SERVER_SIDE: The API is to be called on the database server.
- DB2SEC_CLIENT_SIDE: The API is to be called on a client.

authpluginname

Input. Name of the authentication plug-in that provided the data in the token. The db2secGetGroupsForUser API might use this information in determining the correct group memberships. This parameter might not be filled by DB2 if the authid is not authenticated (for example, if the authid does not match the current connected user).

authpluginnamelen

Input. Length in bytes of the authpluginname parameter value.

grouplist

Output. List of groups to which the user belongs. The list of groups must be returned as a pointer to a section of memory allocated by the plug-in

db2secGetGroupsForUser - Get list of groups for user

containing concatenated varchars (a varchar is a character array in which the first byte indicates the number of bytes following it). The length is an unsigned char (1 byte) and that limits the maximum length of a groupname to 255 characters. For example, "\006GROUP1\007MYGROUP\008MYGROUP3". Each group name should be a valid DB2 authid. The memory for this array must be allocated by the plug-in. The plug-in must therefore provide an API, such as the db2secFreeGroupListMemory API that DB2 will call to free the memory.

numgroups

Output. The number of groups contained in the grouplist parameter.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetGroupsForUser API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Usage notes:

The following is a list of scenarios when problems can occur if an incomplete group list is returned by this API to DB2:

- Embedded SQL application with DYNAMICRULES BIND (or DEFINEDBIND or INVOKEDBIND if the packages are running as a standalone application). DB2 checks for SYSADM membership and the application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- Alternate authorization is provided in CREATE SCHEMA statement. Group lookup will be performed against the AUTHORIZATION NAME parameter if there are nested CREATE statements in the CREATE SCHEMA statement.
- Embedded SQL applications with DYNAMICRULES DEFINERUN/DEFINEBIND and the packages are running in a routine context. DB2 checks for SYSADM membership of the routine definer and the application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- Processing a jar file in an MPP environment. In an MPP environment, the jar processing request is sent from the coordinator node with the session authid. The catalog node received the requests and process the jar files based on the privilege of the session authid (the user executing the jar processing requests).
 - Install jar file. The session authid needs to have one of the following rights: SYSADM, DBADM, or CREATEIN (implicit or explicit on the jar schema). The operation will fail if the above rights are granted to group containing the session authid, but not explicitly to the session authid or if only SYSADM is held, since SYSADM membership is determined by membership in the group defined by a database configuration parameter.
 - Remove jar file. The session authid needs to have one of the following rights: SYSADM, DBADM, or DROPIN (implicit or explicit on the jar schema), or is the definer of the jar file. The operation will fail if the above rights are granted to group containing the session authid, but not explicitly to the session authid, and if the session authid is not the definer of the jar file or if only SYSADM is held since SYSADM membership is determined by membership in the group defined by a database configuration parameter.

db2secGetGroupsForUser - Get list of groups for user

- Replace jar file. This is same as removing the jar file, followed by installing the jar file. Both of the above apply.
- Regenerate views. This is triggered by the ALTER TABLE, ALTER COLUMN, SET DATA TYPE VARCHAR/VARGRAPHIC statements, or during migration. DB2 checks for SYSADM membership of the view definer. The application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- When SET SESSION_USER statement is issued. Subsequent DB2 operations are run under the context of the authid specified by this statement. These operations will fail if the privileges required are owned by one of the SESSION_USER's group is not explicitly granted to the SESSION_USER authid.

Related concepts:

- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for group retrieval plug-ins" on page 627
- "Security plug-in samples" on page 671

db2secDoesGroupExist - Check if group exists

db2secDoesGroupExist - Check if group exists: Determines if an authid represents a group. If the groupname exists, the API must be able to return the value DB2SEC_PLUGIN_OK, to indicate success. It must also be able to return the value DB2SEC_PLUGIN_INVALIDUSERORGROUP if the group name is not valid. It is permissible for the API to return the value DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN if it is impossible to determine if the input is a valid group. If an invalid group (DB2SEC_PLUGIN_INVALIDUSERORGROUP) or group not known (DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN) value is returned, DB2 might not be able to determine whether the authid is a group or user when issuing the GRANT statement without the keywords USER and GROUP, which would result in the error SQLCODE -569, SQLSTATE 56092 being returned to the user.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secDoesGroupExist)
            ( const char *groupname,
              db2int32 groupnamelen,
              char      **errmsg,
              db2int32 *errmsglen );
```

db2secDoesGroupExist API parameters:

groupname

Input. An authid, upper-cased, with no trailing blanks.

groupnamelen

Input. Length in bytes of the groupname parameter value.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesGroupExist API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for group retrieval plug-ins” on page 627
- “Security plug-in samples” on page 671

db2secFreeGroupListMemory - Free group list memory

db2secFreeGroupListMemory - Free group list memory: Frees the memory used to hold the list of groups from a previous call to db2secGetGroupsForUser API.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeGroupListMemory)
( void *ptr,
  char **errmsg,
  db2int32 *errormsglen );
```

db2secFreeGroupListMemory API parameters:

ptr Input. Pointer to the memory to be freed.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeGroupListMemory API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for group retrieval plug-ins” on page 627
- “Security plug-in samples” on page 671

db2secFreeErrorMsg - Free error message memory

db2secFreeErrorMsg - Free error message memory: Frees the memory used to hold an error message from a previous API call. This is the only API that does not return an error message. If this API returns an error, DB2 will log it and continue.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeErrorMsg)
            ( char *errorMsg );
```

db2secFreeErrorMsg API parameters:

msgtofree

Input. A pointer to the error message allocated from a previous API call.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for group retrieval plug-ins” on page 627
- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

User authentication plug-in APIs

APIs for user ID/password authentication plug-ins: For the user ID/password plug-in library, you will need to implement the following client-side APIs:

- db2secClientAuthPluginInit

Note: The db2secClientAuthPluginInit API takes as input a pointer, *logMessage_fn, to an API with the following prototype:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
    db2int32 level,
    void *data,
    db2int32 length
);
```

The db2secLogMessage API allows the plug-in to log messages to db2diag.log for debugging or informational purposes. This API is provided by DB2, so you need not implement it.

- db2secClientAuthPluginTerm
- db2secGenerateInitialCred (Only used for gssapi)
- db2secRemapUserid (Optional)
- db2secGetDefaultLoginContext
- db2secValidatePassword
- db2secProcessServerPrincipalName (This is only for GSS-API)
- db2secFreeToken (Functions to free memory held by the DLL)
- db2secFreeErrorMsg
- db2secFreeInitInfo
- The only API that must be resolvable externally is db2secClientAuthPluginInit. This API will take a void * parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordClientAuthFunctions_1
{
    db2int32 version;
```

APIs for user ID/password authentication plug-ins

```
db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
(
    char        authid[DB2SEC_MAX_AUTHID_LENGTH],
    db2int32    *authidlen,
    char        userid[DB2SEC_MAX_USERID_LENGTH],
    db2int32    *useridlen,
    db2int32    useridtype,
    char        usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
    db2int32    *usernamespacelen,
    db2int32    *usernamespacepacetype,
    const char  *dbname,
    db2int32    dbnamelen,
    void        **token,
    char        **errmsg,
    db2int32    *errmsglen
);
/* Optional */
SQL_API_RC (SQL_API_FN * db2secRemapUserId)
(
    char        userid[DB2SEC_MAX_USERID_LENGTH],
    db2int32    *useridlen,
    char        usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
    db2int32    *usernamespacelen,
    db2int32    *usernamespacepacetype,
    char        password[DB2SEC_MAX_PASSWORD_LENGTH],
    db2int32    *passwordlen,
    char        newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
    db2int32    *newpasswordlen,
    const char  *dbname,
    db2int32    dbnamelen,
    char        **errmsg,
    db2int32    *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secValidatePassword)
(
    const char  *userid,
    db2int32    useridlen,
    const char  *usernamespace,
    db2int32    usernamespacelen,
    db2int32    usernamespacepacetype,
    const char  *password,
    db2int32    passwordlen,
    const char  *newpassword,
    db2int32    newpasswordlen,
    const char  *dbname,
    db2int32    dbnamelen,
    db2Uint32   connection_details,
    void        **token,
    char        **errmsg,
    db2int32    *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
    void        **token,
    char        **errmsg,
    db2int32    *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
    char        *errmsg
);
```

APIs for user ID/password authentication plug-ins

```
SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char    **errmsg,
db2int32 *errmsglen
);
}

or

typedef struct db2secGssapiClientAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
(
char    authid[DB2SEC_MAX_AUTHID_LENGTH],
db2int32 *authidlen,
char    userid[DB2SEC_MAX_USERID_LENGTH],
db2int32 *useridlen,
db2int32 useridtype,
char    usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
db2int32 *usernamespacelen,
db2int32 *usernamespacetype,
const char *dbname,
db2int32 dbnamelen,
void    **token,
char    **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName)
(
const void *data,
gss_name_t *gssName,
char    **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred)
(
const char    *userid,
db2int32    useridlen,
const char    *usernamespace,
db2int32    usernamespaceLen,
db2int32    usernamespaceType,
const char    *password,
db2int32    passwordlen,
const char    *newpassword,
db2int32    newpasswordlen,
const char    *dbname,
db2int32    dbnamelen,
gss_cred_id_t *pGSSCredHandle,
void    **initInfo,
char    **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void    *token,
char    **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
```

APIs for user ID/password authentication plug-ins

```
char *errmsg
);

SQL_API_RC (SQL_API_FN * db2secFreeInitInfo)
(
void      *initInfo,
char      **errmsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char      **errmsg,
db2int32  *errormsglen
);

/* GSS-API specific functions -- refer to db2secPlugin.h
for parameter list*/

OM_uint32 (SQL_API_FN * gss_init_sec_context )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);
}
```

You should use the `db2secUseridPasswordClientAuthFunctions_1` structure if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiClientAuthFunctions_1` structure.

For the user ID/password plug-in library, you will need to implement the following server-side APIs:

- `db2secServerAuthPluginInit`

The `db2secServerAuthPluginInit` API takes as input a pointer, `*logMessage_fn`, to the `db2secLogMessage` API, and a pointer, `*getConDetails_fn`, to the `db2secGetConDetails` API with the following prototypes:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
db2int32 level,
void      *data,
db2int32 length
);

SQL_API_RC (SQL_API_FN db2secGetConDetails)
(
db2int32  conDetailsVersion,
const void *pConDetails
);
```

The `db2secLogMessage` API allows the plug-in to log messages to `db2diag.log` for debugging or informational purposes. The `db2secGetConDetails` API allows the plug-in to obtain details about the client that is trying to attempt to have a database connection. Both the `db2secLogMessage` API and `db2secGetConDetails` API are provided by DB2. So you do not need to implement them. The `db2secGetConDetails` API in turn, takes as its second parameter, `pConDetails`, a pointer to either:

```
typedef struct db2sec_con_details_1
{
db2int32  clientProtocol;
db2uint32 clientIPAddress;
```

APIs for user ID/password authentication plug-ins

```
db2UInt32 connect_info_bitmap;
db2int32  dbnameLen;
char      dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

or

```
typedef struct db2sec_con_details_2
{
    db2int32  clientProtocol;    /* See SQL_PROTOCOL_ in sqlenv.h */
    db2UInt32 clientIPAddress;  /* Set if protocol is TCPIP4    */
    db2UInt32 connect_info_bitmap;
    db2int32  dbnameLen;
    char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
    db2UInt32 clientIP6Address[4]; /* Set if protocol is TCPIP6    */
} db2sec_con_details_2;
```

The possible values for `conDetailsVersion` are `DB2SEC_CON_DETAILS_VERSION_1` and `DB2SEC_CON_DETAILS_VERSION_2` representing the version of the API.

Note: While using `db2sec_con_details_1` or `db2sec_con_details_2`, consider the following:

- Existing plugins that are using the `db2sec_con_details_1` structure and the `DB2SEC_CON_DETAILS_VERSION_1` value will continue to work as they did with v8.2 when calling the `db2GetConDetails` API. If this API is called on an IPv4 platform, the client IP address is returned in the `clientIPAddress` field of the structure. If this API is called on an IPv6 platform, a value of 0 is returned in the `clientIPAddress` field. To retrieve the client IP address on an IPv6 platform, the security plug-in code should be changed to use the `db2sec_con_details_2` structure and the `DB2SEC_CON_DETAILS_VERSION_2` value.
 - New plugins should use the `db2sec_con_details_2` structure and the `DB2SEC_CON_DETAILS_VERSION_2` value. If the `db2secGetConDetails` API is called on an IPv4 platform, the client IP address is returned in the `clientIPAddress` field of the `db2sec_con_details_2` structure and if the API is called on an IPv6 platform the client IP address is returned in the `clientIP6Address` field of the `db2sec_con_details_2` structure. The `clientProtocol` field of the connection details structure will be set to one of `SQL_PROTOCOL_TCPIP` (IPv4, with v1 of the structure), `SQL_PROTOCOL_TCPIP4` (IPv4, with v2 of the structure) or `SQL_PROTOCOL_TCPIP6` (IPv6, with v2 of the structure).
- `db2secServerAuthPluginTerm`
 - `db2secValidatePassword`
 - `db2secGetAuthIDs`
 - `db2secDoesAuthIDExist`
 - `db2secFreeToken`
 - `db2secFreeErrorMsg`
 - The only API that must be resolvable externally is `db2secServerAuthPluginInit`. This API will take a `void *` parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
    db2int32 version;
    db2int32 pluginType;
```

/ parameter lists left blank for readability*

APIs for user ID/password authentication plug-ins

```
    see above for parameters */
SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;

or

typedef struct db2secGssapiServerAuthFunctions_1
{
    db2int32 version;
    db2int32 plugintype;
    gss_buffer_desc serverPrincipalName;
    gss_cred_id_t ServerCredHandle;
    SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)(<parameter list>);
    SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();

    /* GSS-API specific functions
    refer to db2secPlugin.h for parameter list*/
    OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_display_name )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
    OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);

} gssapi_server_auth_functions;
```

You should use the `db2secUseridPasswordServerAuthFunctions_1` structure if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiServerAuthFunctions_1` structure.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related tasks:

- “Deploying a user ID/password plug-in” on page 607

Related reference:

- “db2secClientAuthPluginInit - Initialize client authentication plug-in” on page 646
- “db2secClientAuthPluginTerm - Clean up client authentication plug-in resources” on page 648
- “db2secGetGroupsForUser - Get list of groups for user” on page 633
- “db2secDoesAuthIDExist - Check if authentication ID exists” on page 667
- “db2secFreeErrorMsg - Free error message memory” on page 639
- “db2secFreeInitInfo - Clean up resources held by the db2secGenerateInitialCred” on page 660
- “db2secFreeToken - Free memory held by token” on page 659
- “db2secGenerateInitialCred - Generate initial credentials” on page 653
- “db2secGetAuthIDs - Get authentication IDs” on page 665

APIs for user ID/password authentication plug-ins

- “db2secGetDefaultLoginContext - Get default login context” on page 651
- “db2secProcessServerPrincipalName - Process service principal name returned from server” on page 658
- “db2secRemapUserid - Remap user ID and password” on page 649
- “db2secServerAuthPluginInit - Initialize server authentication plug-in” on page 661
- “db2secServerAuthPluginTerm - Clean up server authentication plug-in resources” on page 664
- “db2secValidatePassword - Validate password” on page 655

db2secClientAuthPluginInit - Initialize client authentication plug-in

db2secClientAuthPluginInit - Initialize client authentication plug-in:

Initialization API, for the client authentication plug-in, that DB2 calls immediately after loading the plug-in.

API and data structure syntax:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit
( db2int32 version,
  void *client_fns,
  db2secLogMessage *logMessage_fn,
  char **errmsg,
  db2int32 *errmsglen );
```

db2secClientAuthPluginInit API parameters:

version

Input. The highest version number of the API that DB2 currently supports. The DB2SEC_API_VERSION value (in db2secPlugin.h) contains the latest version number of the API that DB2 currently supports.

client_fns

Output. A pointer to memory provided by DB2 for a db2secGssapiClientAuthFunctions_<version_number> structure (also known as gssapi_client_auth_functions_<version_number>), if GSS-API authentication is used, or a db2secUseridPasswordClientAuthFunctions_<version_number> structure (also known as userid_password_client_auth_functions_<version_number>), if userid/password authentication is used. The db2secGssapiClientAuthFunctions_<version_number> structure and db2secUseridPasswordClientAuthFunctions_<version_number> structure respectively contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/password authentication plug-in. In future versions of DB2, there might be different versions of the APIs, so the client_fns parameter is cast as a pointer to the gssapi_client_auth_functions_<version_number> structure corresponding to the version the plug-in has implemented.

The first parameter of the gssapi_client_auth_functions_<version_number> structure or the userid_password_client_auth_functions_<version_number> structure tells DB2 the version of the APIs that the plug-in has implemented.

Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented.

Inside the gssapi_server_auth_functions_<version_number> or userid_password_server_auth_functions_<version_number> structure, the plugintype parameter should be set to one of DB2SEC_PLUGIN_TYPE_USERID_PASSWORD, DB2SEC_PLUGIN_TYPE_GSSAPI, or DB2SEC_PLUGIN_TYPE_KERBEROS. Other values can be defined in future versions of the API.

logMessage_fn

Input. A pointer to the db2secLogMessage API, which is implemented by DB2. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to db2diag.log for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag.log file

db2secClientAuthPluginInit - Initialize client authentication plug-in

and the last two parameters respectively are the message string and its length. The valid values for the first parameter of db2secLogMessage API (defined in db2secPlugin.h) are:

- DB2SEC_LOG_NONE (0) No logging
- DB2SEC_LOG_CRITICAL (1) Severe Error encountered
- DB2SEC_LOG_ERROR (2) Error encountered
- DB2SEC_LOG_WARNING (3) Warning
- DB2SEC_LOG_INFO (4) Informational

The message text will show up in the diag.log only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter. So for example, if you use the DB2SEC_LOG_INFO value, the message text will only show up in the db2diag.log if the diaglevel database manager configuration parameter is set to 4.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secClientAuthPluginInit API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 639
- "Security plug-in samples" on page 671
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*

db2secClientAuthPluginTerm - Clean up client authentication plug-in resources

db2secClientAuthPluginTerm - Clean up client authentication plug-in resources: Frees resources used by the client authentication plug-in. This API is called by DB2 just before it unloads the client authentication plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform. For further information, please refer to the topic "Restrictions on security plug-ins".

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secClientAuthPluginTerm)
( char **errmsg,
  db2int32 *errormsglen);
```

db2secClientAuthPluginTerm API parameters:

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secClientAuthPluginTerm API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- "Restrictions on security plug-ins" on page 613
- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 639
- "Security plug-in samples" on page 671

db2secRemapUserid - Remap user ID and password

db2secRemapUserid - Remap user ID and password: This API is called by DB2 on the client side to provide the ability to remap a given user ID and password (and possibly new password and usernamespace) to values different from those given at connect time. DB2 will only call this API if a user ID and a password are supplied at connect time. This prevents a plug-in from remapping a user ID by itself to a user ID/password pair. This API is optional and will not be called if it is not provided or implemented by the security plug-in.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secRemapUserid)
( char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32 *useridlen,
  char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32 *usernamespacelen,
  db2int32 *usernamespacetype,
  char password[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32 *passwordlen,
  char newpasswd[DB2SEC_MAX_PASSWORD_LENGTH],
  db2int32 *newpasswdlen,
  const char *dbname,
  db2int32 dbnameLen,
  char **errmsg,
  db2int32 *errormsglen);
```

db2secRemapUserid API parameters:

userid Input or output. The user ID to be remapped. If there is an input user ID value, then the API must provide an output user ID value that can be the same or different from the input user ID value. If there is no input user ID value, then the API should not return an output user ID value.

useridlen

Input or output. Length in bytes of the userid parameter value.

usernamespace

Input or output. The namespace of the user ID. This value can optionally be remapped. If no input parameter value is specified, but an output value is returned, then the usernamespace will only be used by DB2 for CLIENT type authentication and will be disregarded for other authentication types.

usernamespacelen

Input or output. Length in bytes of the usernamespace parameter value. Under the limitation that the usernamespacetype parameter must be set to the value DB2SEC_NAMESPACE_SAM_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

usernamespacetype

Input or output. Old and new namespace type value. In the current version of DB2, the only supported namespace type value is DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponds to a username style like domain\myname).

password

Input or output. As an input, it is the password that is to be remapped. As an output it is the remapped password. If an input value is specified for this parameter, the API must be able to return an output value that differs from the input value. If no input value is specified, the API must not return an output password value.

passwordlen

Input or output. Length in bytes of the password parameter value.

db2secRemapUserid - Remap user ID and password

newpasswd

Input or output. As an input, it is the new password that is to be set. As an output it is the confirmed new password.

Note: This is the new password that will be passed by DB2 into the newpassword parameter of the db2secValidatePassword API on the client or the server (depending on the value of the authentication database manager configuration parameter). If a new password was passed as input, then the API must be able to return an output value and it can be a different new password. If there is no new password passed in as input, then the API should not return an output new password.

newpasswdlen

Input or output. Length in bytes of the newpasswd parameter value.

dbname

Input. Name of the database to which the client is connecting.

dbnamelen

Input. Length in bytes of the dbname parameter value.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secRemapUserid API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

db2secGetDefaultLoginContext - Get default login context

db2secGetDefaultLoginContext - Get default login context: Determines the user associated with the default login context, in other words, determines the DB2 authid of the user invoking a DB2 command without explicitly specifying a user ID (either an implicit authentication to a database, or a local authorization). This API must return both an authid and a user ID.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGetDefaultLoginContext)
( char authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32 *authidlen,
  char userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32 *useridlen,
  db2int32 useridtype,
  char usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
  db2int32 *usernamespacelen,
  db2int32 *usernamespacetype,
  const char *dbname,
  db2int32 dbnamelen,
  void **token,
  char **errmsg,
  db2int32 *errormsglen );
```

db2secGetDefaultLoginContext API parameters:

authid Output. The parameter in which the authid should be returned. The returned value must conform to DB2 authid naming rules, or the user will not be authorized to perform the requested action.

authidlen

Output. Length in bytes of the authid parameter value.

userid Output. The parameter in which the user ID associated with the default login context should be returned.

useridlen

Output. Length in bytes of the userid parameter value.

useridtype

Input. Indicates if the real or effective user ID of the process is being specified. On Windows, only the real user ID exists. On UNIX and Linux, the real user ID and effective user ID can be different if the uid user ID for the application is different than the ID of the user executing the process. Valid values for the userid parameter (defined in db2secPlugin.h) are:

DB2SEC_PLUGIN_REAL_USER_NAME

Indicates that the real user ID is being specified.

DB2SEC_PLUGIN_EFFECTIVE_USER_NAME

Indicates that the effective user ID is being specified.

Note: Some plug-in implementations might not distinguish between the real and effective userid. In particular, a plug-in that does not use the UNIX or Linux identity of the user to establish the DB2 AUTHID can safely ignore this distinction.

usernamespace

Output. The namespace of the user ID.

usernamespacelen

Output. Length in bytes of the usernamespace parameter value. Under the limitation that the usernamespacetype parameter must be set to the value

db2secGetDefaultLoginContext - Get default login context

DB2SEC_NAMESPACE_SAM_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

usernamespace

Output. Namespace type value. In the current version of DB2, the only supported namespace type is DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponds to a username style like domain\myname).

dbname

Input. Contains the name of the database being connected to, if this call is being used in the context of a database connection. For local authorization actions or instance attachments, this parameter is set to NULL.

dbnamelen

Input. Length in bytes of the dbname parameter value.

token Output. This is a pointer to data allocated by the plug-in that it might pass to subsequent authentication calls in the plug-in, or possibly to the group retrieval plug-in. The structure of this data is determined by the plug-in writer.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetDefaultLoginContext API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671
- “authentication - Authentication type configuration parameter” in *Performance Guide*

db2secGenerateInitialCred - Generate initial credentials

db2secGenerateInitialCred - Generate initial credentials: Obtains the initial GSS-API credentials based on the user ID and password that are passed in. For Kerberos, this will be the ticket-granting ticket (TGT). The credential handle that is returned in `pGSSCredHandle` parameter is the handle that is used with the `gss_init_sec_context` API and must be either an INITIATE or BOTH credential. The `db2secGenerateInitialCred` API is only called when a user ID, and possibly a password are supplied. Otherwise, DB2 will specify the value `GSS_C_NO_CREDENTIAL` when calling the `gss_init_sec_context` API to signify that the default credential obtained from the current login context is to be used.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGenerateInitialCred)
( const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32 usernamespacectype,
  const char *password,
  db2int32 passwordlen,
  const char *newpassword,
  db2int32 newpasswordlen,
  const char *dbname,
  db2int32 dbnamehlen,
  gss_cred_id_t *pGSSCredHandle,
  void **InitInfo,
  char **errormsg,
  db2int32 *errormsglen );
```

db2secGenerateInitialCred API parameters:

userid Input. The user ID whose password is to be verified on the database server.

useridlen

Input. Length in bytes of the `userid` parameter value.

usernamespace

Input. The namespace from which the user ID was obtained.

usernamespacehlen

Input. Length in bytes of the `usernamespace` parameter value.

usernamespacectype

Input. The type of namespace.

password

Input. The password to be verified.

passwordlen

Input. Length in bytes of the `password` parameter value.

newpassword

Input. A new password if the password is to be changed. If no change is requested, the `newpassword` parameter is set to NULL. If it is not NULL, the API should validate the old password before setting the password to its new value. The API does not have to honour a request to change the password, but if it does not, it should immediately return with the return value `DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED` without validating the old password.

newpasswordlen

Input. Length in bytes of the `newpassword` parameter value.

db2secGenerateInitialCred - Generate initial credentials

dbname

Input. The name of the database being connected to. The API is free to ignore this parameter, or the API can return the value DB2SEC_PLUGIN_CONNECTION_DISALLOWED if it has a policy of restricting access to certain databases to users who otherwise have valid passwords.

dbnamelen

Input. Length in bytes of the dbname parameter value.

pGSSCredHandle

Output. Pointer to the GSS-API credential handle.

InitInfo

Output. A pointer to data that is not known to DB2. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. DB2 will call the db2secFreeInitInfo API at the end of the authentication process, at which point these resources are freed. If the db2secGenerateInitialCred API does not need to maintain such a list, then it should return NULL.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGenerateInitialCred API execution is not successful.

Note: For this API, error messages should not be created if the return value indicates a bad user ID or password. An error message should only be returned if there is an internal error in the API that prevented it from completing properly.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592
- “Authentication methods for your server” in *Administration Guide: Implementation*

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

db2secValidatePassword - Validate password

db2secValidatePassword - Validate password: Provides a method for performing user ID and password style authentication during a database connect operation.

Note: When the API is run on the client side, the API code is run with the privileges of the user executing the CONNECT statement. This API will only be called on the client side if the authentication configuration parameter is set to CLIENT.

When the API is run on the server side, the API code is run with the privileges of the instance owner.

The plug-in writer should take the above into consideration if authentication requires special privileges (such as root level system access on UNIX).

This API must return the value DB2SEC_PLUGIN_OK (success) if the password is valid, or an error code such as DB2SEC_PLUGIN_BADPWD if the password is invalid.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secValidatePassword)
( const char *userid,
  db2int32 useridlen,
  const char *usernamespace,
  db2int32 usernamespacelen,
  db2int32 usernamespacectype,
  const char *password,
  db2int32 passwordlen,
  const char *newpasswd,
  db2int32 newpasswdlen,
  const char *dbname,
  db2int32 dbnameLen,
  db2UInt32 connection_details,
  void      **token,
  char      **errmsg,
  db2int32 *errmsglen );
```

db2secValidatePassword API parameters:

userid Input. The user ID whose password is to be verified.

useridlen

Input. Length in bytes of the userid parameter value.

usernamespace

Input. The namespace from which the user ID was obtained.

usernamespaceLen

Input. Length in bytes of the usernamespace parameter value.

usernamespacectype

Input. The type of namespace. Valid values for the usernamespacectype parameter (defined in db2secPlugin.h) are:

- DB2SEC_NAMESPACE_SAM_COMPATIBLE Corresponds to a username style like domain\myname
- DB2SEC_NAMESPACE_USER_PRINCIPAL Corresponds to a username style like myname@domain.ibm.com

Currently DB2 only supports the value DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, the usernamespacectype parameter is set to the value DB2SEC_USER_NAMESPACE_UNDEFINED (defined in db2secPlugin.h).

db2secValidatePassword - Validate password

password

Input. The password to be verified.

passwordlen

Input. Length in bytes of the password parameter value.

newpasswd

Input. A new password, if the password is to be changed. If no change is requested, this parameter is set to NULL. If this parameter is not NULL, the API should validate the old password before changing it to the new password. The API does not have to fulfill a request to change the password, but if it does not, it should immediately return with the return value `DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED` without validating the old password.

newpasswdlen

Input. Length in bytes of the newpasswd parameter value.

dbname

Input. The name of the database being connected to. The API is free to ignore the dbname parameter, or it can return the value `DB2SEC_PLUGIN_CONNECTIONREFUSED` if it has a policy of restricting access to certain databases to users who otherwise have valid passwords. This parameter can be NULL.

dbnamelen

Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL.

connection_details

Input. A 32-bit parameter of which 3 bits are currently used to store the following information:

- The right-most bit indicates whether the source of the user ID is the default from the `db2secGetDefaultLoginContext` API, or was explicitly provided during the connect.
- The second-from-right bit indicates whether the connection is local (using Inter Process Communication (IPC) or a connect from one of the nodes in the `db2nodes.cfg` in the partitioned database environment), or remote (through a network or loopback). This gives the API the ability to decide whether clients on the same machine can connect to the DB2 server without a password. Due to the default OS-based user ID/password plugin, local connections are permitted without a password from clients on the same machine (assuming the user has connect privileges).
- The third-from-right bit indicates whether DB2 is calling the API from the server side or client side.

The bit values are defined in `db2secPlugin.h`:

- `DB2SEC_USERID_FROM_OS` (0x00000001) Indicates that the user ID is obtained from OS and not explicitly given on the connect statement.
- `DB2SEC_CONNECTION_ISLOCAL` (0x00000002) Indicates a local connection.
- `DB2SEC_VALIDATING_ON_SERVER_SIDE` (0x00000004) Indicates whether DB2 is calling from the server side or client side to validate password. If this bit value is set, then DB2 is calling from server side, otherwise it is calling from the client side.

DB2's default behavior for an implicit authentication is to allow the connection without any password validation. However, plug-in developers

db2secValidatePassword - Validate password

have the option to disallow implicit authentication by returning a DB2SEC_PLUGIN_BADPASSWORD error.

token Input. A pointer to data which can be passed as a parameter to subsequent API calls during the current connection. Possible APIs that might be called include db2secGetAuthIDs API and db2secGetGroupsForUser API.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secValidatePassword API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “Security plug-in samples” on page 671
- “APIs for user ID/password authentication plug-ins” on page 639

db2secProcessServerPrincipalName - Process service principal name returned from server

db2secProcessServerPrincipalName - Process service principal name returned from server: Processes the service principal name returned from the server and returns the principal name in the gss_name_t internal format to be used with the gss_init_sec_context API. The db2secProcessServerPrincipalName API also processes the service principal name cataloged with the database directory when Kerberos authentication is used. Ordinarily, this conversion uses the gss_import_name API. After the context is established, the gss_name_t object is freed through the call to gss_release_name API. The db2secProcessServerPrincipalName API returns the value DB2SEC_PLUGIN_OK if gssName parameter points to a valid GSS name; a DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME error code is returned if the principal name is invalid.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secProcessServerPrincipalName)
( const char *name,
  db2int32 namelen,
  gss_name_t *gssName,
  char      **errorMsg,
  db2int32 *errormsglen );
```

db2secProcessServerPrincipalName API parameters:

name Input. Text name of the service principal in GSS_C_NT_USER_NAME format; for example, service/host@REALM.

namelen

Input. Length in bytes of the name parameter value.

gssName

Output. Pointer to the output service principal name in the GSS-API internal format.

errorMsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secProcessServerPrincipalName API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errorMsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

db2secFreeToken - Free memory held by token

db2secFreeToken - Free memory held by token: Frees the memory held by token. This API is called by DB2 when it no longer needs the memory held by the token parameter.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeToken)
            ( void *token,
              char      **errmsg,
              db2int32 *errormsglen );
```

db2secFreeToken API parameters:

token Input. Pointer to the memory to be freed.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeToken API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

db2secFreeInitInfo - Clean up resources held by the db2secGenerateInitialCred

db2secFreeInitInfo - Clean up resources held by the db2secGenerateInitialCred:

Frees any resources allocated by the db2secGenerateInitialCred API. This can include, for example, handles to underlying mechanism contexts or a credential cache created for the GSS-API credential cache.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeInitInfo)
( void *initinfo,
  char **errmsg,
  db2int32 *errormsglen);
```

db2secFreeInitInfo API parameters:

initinfo

Input. A pointer to data that is not known to DB2. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. These resources are freed by calling this API.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeInitInfo API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

db2secServerAuthPluginInit - Initialize server authentication plug-in

db2secServerAuthPluginInit - Initialize server authentication plug-in:

Initialization API, for the server authentication plug-in, that DB2 calls immediately after loading the plug-in. In the case of GSS-API, the plug-in is responsible for filling in the server's principal name in the serverPrincipalName parameter inside the gssapi_server_auth_functions structure at initialization time and providing the server's credential handle in the serverCredHandle parameter inside the gssapi_server_auth_functions structure. The freeing of the memory allocated to hold the principal name and the credential handle must be done by the db2secServerAuthPluginTerm API by calling the gss_release_name and gss_release_cred APIs.

API and data structure syntax:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit
( db2int32 version,
  void *server_fns,
  db2secGetConDetails *getConDetails_fn,
  db2secLogMessage *logMessage_fn,
  char **errorMsg,
  db2int32 *errormsglen );
```

db2secServerAuthPluginInit API parameters:

version

Input. The highest version number of the API that DB2 currently supports. The DB2SEC_API_VERSION value (in db2secPlugin.h) contains the latest version number of the API that DB2 currently supports.

server_fns

Output. A pointer to memory provided by DB2 for a db2secGssapiServerAuthFunctions_<version_number> structure (also known as gssapi_server_auth_functions_<version_number>), if GSS-API authentication is used, or a db2secUseridPasswordServerAuthFunctions_<version_number> structure (also known as userid_password_server_auth_functions_<version_number>), if userid/password authentication is used. The db2secGssapiServerAuthFunctions_<version_number> structure and db2secUseridPasswordServerAuthFunctions_<version_number> structure respectively contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/password authentication plug-in.

In future versions of DB2, there might be different versions of the APIs, so the server_fns parameter is cast as a pointer to the gssapi_server_auth_functions_<version_number> structure corresponding to the version the plug-in has implemented. The first parameter of the gssapi_server_auth_functions_<version_number> structure or the userid_password_server_auth_functions_<version_number> structure tells DB2 the version of the APIs that the plug-in has implemented.

Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented.

Inside the gssapi_server_auth_functions_<version_number> or userid_password_server_auth_functions_<version_number> structure, the plugintype parameter should be set to one of DB2SEC_PLUGIN_TYPE_USERID_PASSWORD, DB2SEC_PLUGIN_TYPE_GSSAPI, or DB2SEC_PLUGIN_TYPE_KERBEROS. Other values can be defined in future versions of the API.

db2secServerAuthPluginInit - Initialize server authentication plug-in

getConDetails_fn

Input. Pointer to the db2secGetConDetails API, which is implemented by DB2. The db2secServerAuthPluginInit API can call the db2secGetConDetails API in any one of the other authentication APIs to obtain details related to the database connection. These details include information about the communication mechanism associated with the connection (such as the IP address, in the case of TCP/IP), which the plug-in writer might need to reference when making authentication decisions. For example, the plug-in could disallow a connection for a particular user, unless that user is connecting from a particular IP address. The use of the db2secGetConDetails API is optional.

If the db2secGetConDetails API is called in a situation not involving a database connection, it returns the value DB2SEC_PLUGIN_NO_CON_DETAILS, otherwise, it returns 0 on success.

The db2secGetConDetails API takes two input parameters; pConDetails, which is a pointer to the db2sec_con_details_<version_number> structure, and conDetailsVersion, which is a version number indicating which db2sec_con_details structure to use. Possible values are DB2SEC_CON_DETAILS_VERSION_1 when db2sec_con_details1 is used or DB2SEC_CON_DETAILS_VERSION_2 when db2sec_con_details2. The recommended version number to use is DB2SEC_CON_DETAILS_VERSION_2.

Upon a successful return, the db2sec_con_details structure (either db2sec_con_details1 or db2sec_con_details2) will contain the following information:

- The protocol used for the connection to the server. The listing of protocol definitions can be found in the file sqlenv.h (located in the include directory) (SQL_PROTOCOL_*). This information is filled out in the clientProtocol parameter.
- The TCP/IP address of the inbound connect to the server if the clientProtocol is SQL_PROTOCOL_TCPIP or SQL_PROTOCOL_TCPIP4. This information is filled out in the clientIPAddress parameter.
- The database name the client is attempting to connect to. This will not be set for instance attachments. This information is filled out in the dbname and dbnameLen parameters.
- A connection information bit-map that contains the same details as documented in the connection_details parameter of the db2secValidatePassword API. This information is filled out in the connect_info_bitmap parameter.
- The TCP/IP address of the inbound connect to the server if the clientProtocol is SQL_PROTOCOL_TCPIP6. This information is filled out in the clientIP6Address parameter and it is only available if DB2SEC_CON_DETAILS_VERSION_2 is used for db2secGetConDetails API call.

logMessage_fn

Input. A pointer to the db2secLogMessage API, which is implemented by DB2. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to db2diag.log for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag.log file

db2secServerAuthPluginInit - Initialize server authentication plug-in

and the last two parameters respectively are the message string and its length. The valid values for the first parameter of dbsecLogMessage API (defined in db2secPlugin.h) are:

DB2SEC_LOG_NONE (0)

No logging

DB2SEC_LOG_CRITICAL (1)

Severe Error encountered

DB2SEC_LOG_ERROR (2)

Error encountered

DB2SEC_LOG_WARNING (3)

Warning

DB2SEC_LOG_INFO (4)

Informational

The message text will show up in the diag.log only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter.

So for example, if you use the DB2SEC_LOG_INFO value, the message text will only show up in the db2diag.log if the diaglevel database manager configuration parameter is set to 4.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secServerAuthPluginInit API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 639
- "Security plug-in samples" on page 671
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*

db2secServerAuthPluginTerm - Clean up server authentication plug-in resources

db2secServerAuthPluginTerm - Clean up server authentication plug-in resources: Frees resources used by the server authentication plug-in. This API is called by DB2 just before it unloads the server authentication plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform. For further information, please refer to the topic "Restrictions on security plug-ins".

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secServerAuthPluginTerm)
( char **errmsg,
  db2int32 *errormsglen );
```

db2secServerAuthPluginTerm API parameters:

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secServerAuthPluginTerm API execution is not successful.

errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errmsg parameter.

Related concepts:

- "Restrictions on security plug-ins" on page 613
- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 639
- "Security plug-in samples" on page 671

db2secGetAuthIDs - Get authentication IDs: Returns an SQL authid for an authenticated user. This API is called during database connections for both user ID/password and GSS-API authentication methods.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGetAuthIDs)
    ( const char *userid,
      db2int32 useridlen,
      const char *usernamespace,
      db2int32 usernamespacelen,
      db2int32 usernamespacectype,
      const char *dbname,
      db2int32 dbnamelen,
      void      **token,
      char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
      db2int32 *SystemAuthIDlen,
      char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
      db2int32 *InitialSessionAuthIDlen,
      char username[DB2SEC_MAX_USERID_LENGTH],
      db2int32 *usernamelen,
      db2int32 *initsessionidtype,
      char      **errmsg,
      db2int32 *errormsglen );
```

db2secGetAuthIDs API parameters:

userid Input. The authenticated user. This is blank for GSS-API authentication method.

useridlen

Input. Length in bytes of the userid parameter value.

usernamespace

Input. The namespace from which the user ID was obtained.

usernamespacecctype

Input. Length in bytes of the usernamespace parameter value.

usernamespacectype

Input. Namespace type value. In the current version of DB2, the only supported namespace type value is DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponds to a username style like domain\myname).

dbname

Input. The name of the database being connected to. The API can ignore this, or it can return differing authids when the same user connects to different databases. This parameter can be NULL.

dbnamelen

Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL.

token

Input or output. Data that the plug-in might pass to the db2secGetGroupsForUser API. For GSS-API, this is a context handle (gss_ctx_id_t). Ordinarily, token is an input-only parameter and its value is taken from the db2secValidatePassword API. It can also be an output parameter when authentication is done on the client and therefore db2secValidatePassword API is not called.

db2secGetAuthIDs - Get authentication IDs

SystemAuthID

Output. The system authid that corresponds to the ID of the authenticated user. The size is 255 bytes, but DB2 currently uses only up to (and including) 30 bytes.

SystemAuthIDlen

Output. Length in bytes of the SystemAuthID parameter value.

InitialSessionAuthID

Output. Authid used for this connection session. This is usually the same as the SystemAuthID parameter but can be different in some situations, for instance, when issuing a SET SESSION AUTHORIZATION statement. The size is 255 bytes, but DB2 currently uses only up to (and including) 30 bytes.

InitialSessionAuthIDlen

Output. Length in bytes of the InitialSessionAuthID parameter value.

username

Output. A username corresponding to the authenticated user and authid. This will only be used for auditing and will be logged in the "User ID" field in the audit record for CONNECT statement. If the API does not fill in the username parameter, DB2 copies it from the userid.

usernameLen

Output. Length in bytes of the username parameter value.

initSessionIDType

Output. Session authid type indicating whether or not the InitialSessionAuthid parameter is a role or an authid. The API should return one of the following values (defined in db2secPlugin.h):

- DB2SEC_ID_TYPE_AUTHID (0)
- DB2SEC_ID_TYPE_ROLE (1)

Currently, DB2 only supports authid (DB2SEC_ID_TYPE_AUTHID).

errorMsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetAuthIDs API execution is not successful.

errorMsgLen

Output. A pointer to an integer that indicates the length in bytes of the error message string in errorMsg parameter.

Related concepts:

- "Security plug-in APIs" on page 625
- "Security plug-ins" on page 592

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 639
- "Security plug-in samples" on page 671

db2secDoesAuthIDExist - Check if authentication ID exists

db2secDoesAuthIDExist - Check if authentication ID exists: Determines if the authid represents an individual user (for instance, whether the API can map the authid to an external user id). The API should return the value DB2SEC_PLUGIN_OK if it is successful - the authid is valid, DB2SEC_PLUGIN_INVALID_USERORGROUP if it is not valid, or DB2SEC_PLUGIN_USERSTATUSNOTKNOWN if the authid existence cannot be determined.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secDoesAuthIDExist)
            ( const char *authid,
              db2int32 authidlen,
              char      **errmsg,
              db2int32 *errmsglen );
```

db2secDoesAuthIDExist API parameters:

authid Input. The authid to validate. This is upper-cased, with no trailing blanks.

authidlen

Input. Length in bytes of the authid parameter value.

errmsg

Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesAuthIDExist API execution is not successful.

errmsglen

Output. A pointer to an integer that indicates the length of the error message string in errmsg parameter.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 639
- “Security plug-in samples” on page 671

GSS-API plug-in APIs

Required APIs and definitions for GSS-API authentication plug-ins: Following is a complete list of GSS-APIs required for the DB2 security plug-in interface. The supported APIs follow these specifications: *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Before implementing a GSS-API based plug-in, you should have a complete understanding of these specifications.

Table 83. Required APIs and Definitions for GSS-API authentication plug-ins

Name		Description
Client-side APIs	gss_init_sec_context	Initiate a security context with a peer application.
Server-side APIs	gss_accept_sec_context	Accept a security context initiated by a peer application.
Server-side APIs	gss_display_name	Convert an internal format name to text.
Common APIs	gss_delete_sec_context	Delete an established security context.
Common APIs	gss_display_status	Obtain the text error message associated with a GSS-API status code.

Required APIs and definitions for GSS-API authentication plug-ins

Table 83. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

Name		Description
Common APIs	gss_release_buffer	Delete a buffer.
Common APIs	gss_release_cred	Release local data structures associated with a GSS-API credential.
Common APIs	gss_release_name	Delete internal format name.
Required definitions	GSS_C_DELEG_FLAG	Requests delegation.
Required definitions	GSS_C_EMPTY_BUFFER	Signifies that the gss_buffer_desc does not contain any data.
Required definitions	GSS_C_GSS_CODE	Indicates a GSS major status code.
Required definitions	GSS_C_INDEFINITE	Indicates that the mechanism does not support context expiration.
Required definitions	GSS_C_MECH_CODE	Indicates a GSS minor status code.
Required definitions	GSS_C_MUTUAL_FLAG	Mutual authentication requested.
Required definitions	GSS_C_NO_BUFFER	Signifies that the gss_buffer_t variable does not point to a valid gss_buffer_desc structure.
Required definitions	GSS_C_NO_CHANNEL_BINDINGS	No communication channel bindings.
Required definitions	GSS_C_NO_CONTEXT	Signifies that the gss_ctx_id_t variable does not point to a valid context.
Required definitions	GSS_C_NO_CREDENTIAL	Signifies that gss_cred_id_t variable does not point to a valid credential handle.
Required definitions	GSS_C_NO_NAME	Signifies that the gss_name_t variable does not point to a valid internal name.
Required definitions	GSS_C_NO_OID	Use default authentication mechanism.
Required definitions	GSS_C_NULL_OID_SET	Use default mechanism.
Required definitions	GSS_S_COMPLETE	API completed successfully.
Required definitions	GSS_S_CONTINUE_NEEDED	Processing is not complete and the API must be called again with the reply token received from the peer.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “Restrictions for GSS-API authentication plug-ins” on page 669

Restrictions for GSS-API authentication plug-ins

Restrictions for GSS-API authentication plug-ins: The following is a list of restrictions for GSS-API authentication plug-ins.

- The default security mechanism is always assumed; therefore, there is no OID consideration.
- The only GSS services requested in `gss_init_sec_context()` are mutual authentication and delegation. DB2 always requests a ticket for delegation, but does not use that ticket to generate a new ticket.
- Only the default context time is requested.
- Context tokens from `gss_delete_sec_context()` are not sent from the client to the server and vice-versa.
- Anonymity is not supported.
- Channel binding is not supported
- If the initial credentials expire, DB2 does not automatically renew them.
- The GSS-API specification stipulates that even if `gss_init_sec_context()` or `gss_accept_sec_context()` fail, either function must return a token to send to the peer. However, because of DRDA limitations, DB2 only sends a token if `gss_init_sec_context()` fails and generates a token on the first call.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Related reference:

- “Required APIs and definitions for GSS-API authentication plug-ins” on page 667

Security plug-in API versioning

Because it is possible that future releases of DB2 will need different versions of the security plug-in APIs, DB2 supports version numbering of the APIs. These version numbers are integers starting with 1 for DB2 UDB version 8.2. The version number that DB2 passes to the security plug-in APIs are the highest version number of the API that DB2 can support, and correspond to the version number of the structure. If the plug-in can support a higher API version, it must return function pointers for the version that DB2 has requested. If the plug-in only supports a lower version of the API, the plug-in should fill in function pointers for the lower version. In either situation, the security plug-in APIs should return the version number for the API it is supporting in the version field of the functions structure.

For DB2, the version numbers of the security plug-ins will only change when necessary (for example, when there are changes to the parameters of the APIs). Version numbers will not automatically change with DB2 release numbers.

Related concepts:

- “Security plug-in APIs” on page 625
- “Security plug-ins” on page 592

Security plug-in samples

UNIX directory: The 'C' samples are located in `sqllib/samples/security/plugins` and the JCC GSS-API plugin samples (.java) are located in `sqllib/samples/java/jdbc`

Windows directory: The 'C' samples are located in `sqllib\samples\security\plugins` and the JCC GSS-API plugin samples (.java) are located in `sqllib\samples\java\jdbc`

Table 84. Security plug-in sample program files

Sample program name	Program description
<code>combined.c</code>	Combined userid/password authentication and group lookup sample.
<code>group_file.c</code>	Simple file-based group management plug-in sample.
<code>gssapi_simple.c</code>	Basic GSS-API authentication plug-in sample (both client and server).
<code>IBMkrb5.c</code>	Source code for the IBM-supplied Kerberos security plug-in for UNIX.
<code>JCCKerberosPlugin.java</code>	Implement a GSS-API Plugin that does Kerberos authentication using IBM DB2 Universal Driver.
<code>JCCKerberosPluginTest.java</code>	Use <code>JCCKerberosPlugin</code> to get a DB2 Connection using IBM DB2 Universal Driver.
<code>JCCSimpleGSSPlugin.java</code>	Implement a GSS-API Plugin that does userid and password checking using IBM DB2 Universal Driver.
<code>JCCSimpleGSSContext.java</code>	Implement a <code>GSSContext</code> to be used by <code>JCCSimpleGSSPlugin</code>
<code>JCCSimpleGSSCredential.java</code>	Implement a <code>GSSCredential</code> to be used by <code>JCCSimpleGSSPlugin</code>
<code>JCCSimpleGSSException.java</code>	Implement a <code>GSSException</code> to be used by <code>JCCSimpleGSSPlugin</code>
<code>JCCSimpleGSSName.java</code>	Implement a <code>GSSName</code> to be used by <code>JCCSimpleGSSPlugin</code>
<code>JCCSimpleGSSPluginTest.java</code>	Use <code>JCCSimpleGSSPlugin</code> to get a DB2 Connection using IBM DB2 Universal Driver.

Related concepts:

- "Security plug-ins" on page 592
- "Sample files" in *Samples Topics*

Backup and restore plug-ins for vendor storage devices

DB2 APIs for backup and restore to storage managers

DB2 provides an interface that can be used by third-party media management products to store and retrieve data for backup and restore operations and log files. This interface is designed to augment the backup, restore, and log archiving data targets of diskette, disk, tape, and Tivoli Storage Manager, that are supported as a standard part of DB2.

APIs for backup and restore to storage managers

These third-party media management products will be referred to as vendor products in the remainder of this section.

DB2 defines a set of API prototypes that provide a general purpose data interface to backup, restore, and log archiving that can be used by many vendors. These APIs are to be provided by the vendor in a shared library on UNIX based systems, or DLL on the Windows operating system. When the APIs are invoked by DB2, the shared library or DLL specified by the calling backup, restore, or log archiving routine is loaded and the APIs provided by the vendor are called to perform the required tasks.

Sample files demonstrating the DB2 vendor functionality are located on UNIX platforms in the `sql1lib/samples/BARVendor` directory, and on Windows in the `sql1lib\samples\BARVendor` directory.

The following are the definitions for terminology used in the descriptions of the backup and restore vendor storage plug-in APIs.

Backup and restore vendor storage plug-in

A dynamically loadable library that DB2 will load to access user-written backup and restore APIs for vendor products.

Input Indicates that DB2 will fill in the value for the backup and restore vendor storage plug-in API parameter.

Output

Indicates that the backup and restore vendor storage plug-in API will fill in the value for the API parameter.

Operational overview

Seven APIs are defined to provide a data interface between DB2 and the vendor product:

- `sqluvint` - Initialize and link to a vendor device
- `sqluvget` - Read data from a vendor device
- `sqluvput` - Write data to a vendor device
- `sqluvend` - Unlink the device and release its resources
- `sqluvdel` - Delete committed session
- `db2VendorQueryApiVersion` - Query device supported API level
- `db2VendorGetNextObj` - Get next object on device

DB2 will call these APIs, 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 API may compromise data integrity of the database.

The sequence of APIs 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, a restore, a log archive, or a log retrieve operation.
- The PROMPTING mode that is specified on the backup or restore operation.
- The characteristics of the vendor 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 APIs 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 APIs.

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 the sqluvint API.

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 backing up an image to a vendor device, the following sequence of calls is issued by DB2 for *each* session:

```
db2VendorQueryApiVersion
```

followed by 1

```
sqluvint, action = SQLUV_WRITE
```

followed by 1 to n

```
sqluvput
```

followed by 1

```
sqluwend, action = SQLUV_COMMIT
```

When DB2 issues an sqluwend 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.

APIs for backup and restore to storage managers

Note: For backing up a log file to a vendor device, use action = SQLUV_ARCHIVE with the sqluvint call.

For restoring an image from a vendor device, the sequence of calls for each session is:

```
db2VendorQueryApiVersion
```

followed by 1

```
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: For restoring a log file from a vendor device, use action = SQLUV_RETRIEVE with the sqluvint call.

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.

For searching a vendor device for an image or archived log file, the following sequence of calls is issued by DB2 for each session:

```
sqluvint, action = SQLUV_QUERY_IMAGES or SQLUV_QUERY_LOGS
```

followed by 1 to n

```
db2VendorGetNextObj
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

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 sqluwend (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 restore operation completes.
- If the response is *terminate*, DB2 terminates the backup or restore operation. For more information on exactly what DB2 does to terminate the sessions, see “If error conditions are returned to DB2” on page 676.

APIs for backup and restore to storage managers

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 `sqlvdel` 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.

APIs for backup and restore to storage managers

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 0. 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” on page 678.

APIs for backup and restore to storage managers

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 <i>Use Library</i> . Specify the library name (on UNIX based systems) or the DLL name (on the Windows operating system).
Number of sessions	Is <i>Sessions</i> .
Vendor options	Is not supported.
Vendor file name	Is not supported.
Transfer buffer size	Is (for backup) <i>Size of each Buffer</i> , 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	<i>library-name</i>	<i>shared-library</i>
Number of sessions	<i>num-sessions</i>	<i>num-sessions</i>
Vendor options	<i>options-string</i>	<i>options-string</i>
Vendor file name	<i>file-name</i>	<i>file-name</i>
Transfer buffer size	<i>buffer-size</i>	<i>buffer-size</i>

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 <i>sqlu_media_list</i> , specify a media type of SQLU_OTHER_MEDIA, and then in structure <i>sqlu_vendor</i> , specify a shared library or DLL in <i>shr_lib</i> .

APIs for backup and restore to storage managers

To specify	The API parameter (for both db2Backup and db2Restore) is
Number of sessions	as follows: In structure <i>sqlu_media_list</i> , specify <i>sessions</i> .
Vendor options	<i>PVendorOptions</i>
Vendor file name	as follows: In structure <i>sqlu_media_list</i> , specify a media type of <i>SQLU_OTHER_MEDIA</i> , and then in structure <i>sqlu_vendor</i> , specify a file name in <i>filename</i> .
Transfer buffer size	<i>BufferSize</i>

Related concepts:

- “DB2 database system plug-ins for customizing database management” on page 591

Related reference:

- “db2VendorGetNextObj - Get next object on device” on page 694
- “db2VendorQueryApiVersion - Get the supported level of the vendor storage API” on page 693
- “Data ” on page 704
- “DB2_info ” on page 696
- “Init_input ” on page 701
- “Init_output ” on page 703
- “Return_code ” on page 705
- “sqluvdel - Delete committed session” on page 691
- “sqluvend - Unlink a vendor device and release its resources” on page 689
- “sqluvget - Read data from a vendor device” on page 685
- “sqluvint - Initialize and link to a vendor device” on page 680
- “sqluvput - Write data to a vendor device” on page 687
- “Vendor_info ” on page 699

sqluvint - Initialize and link to a vendor device

Provides information for initializing a vendor device and for establishing a logical link between DB2 and the vendor device.

Authorization:

None

Required connection:

Database

API include file:

sqluvend.h

API and data structure syntax:

```
int sqluvint ( struct Init_input *in,  
              struct Init_output *out,  
              struct Return_code *return_code);
```

sqluvint API parameters:

in Input. Structure that contains information provided by DB2 to establish a logical link with the vendor device.

out 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 API to obtain a device handle. If for any reason, the vendor storage API 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 API. Details on possible return codes, and the DB2 reaction to each of these, is contained in the return codes table (see table below).

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

sqluvint - Initialize and link to a vendor device

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 calling other data transfer APIs, but may terminate the session at any time with an sqluvend call.

Return codes:

Table 85. 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 to the sqluvput API (to BACKUP data). If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_OK; the next call will be to the sqluvget API 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 API 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 API call will not be received, since the session was never established.

sqluvint - Initialize and link to a vendor device

Table 85. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
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 API 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 API 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 API 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 API 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 API call will not be received, since the session was never established.

sqluvint - Initialize and link to a vendor device

Table 85. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_INV_USERID	Invalid userid specified.	No further calls.	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend API 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 API call will not be received, since the session was never established.
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 API 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 API 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 API call will not be received, since the session was never established.

sqluvint - Initialize and link to a vendor device

Table 85. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
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 to sqluvput API. If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_MAX_LINK_GRANT; the next call will be to the sqluvget API 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 API 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 API call will not be received, since the session was never established.

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671
- “Init_input ” on page 701
- “Init_output ” on page 703
- “Return_code ” on page 705

sqluvget - Read data from a vendor device

After a vendor device has been initialized with the sqluvint API, DB2 calls this API to read from the device during a restore operation.

Authorization:

None

Required connection:

Database

API include file:

sqluvend.h

API and data structure syntax:

```
int sqluvget ( void * hdl,
              struct Data *data,
              struct Return_code *return_code);
```

sqluvget API parameters:

hdl Input. Pointer to space allocated for the Data structure (including the data buffer) and Return_code.

data Input or output. A pointer to the Data structure.

return_code

Output. The return code from the API call.

Usage notes:

This API is used by the restore utility.

Return codes:

Table 86. 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 (see note below)	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT (see note below)	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT (see note below)	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT (see note below)	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT (see note below)	The session will be terminated.

sqluvget - Read data from a vendor device

Table 86. Valid Return Codes for sqluvget and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
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 (see note below)	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 (see note below)	The session will be terminated.

Note: Next call: If the next call is an sqluvend, action = SQLU_ABORT, this session and all other active sessions will be terminated.

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671
- “Data ” on page 704
- “Return_code ” on page 705

sqluvput - Write data to a vendor device

After a vendor device has been initialized with the sqluvint API, DB2 calls this API to write to the device during a backup operation.

Authorization:

None

Required connection:

Database

API include file:

sqluvend.h

API and data structure syntax:

```
int sqluvput ( void *          hdl,
              struct Data *data,
              struct Return_code *return_code);
```

sqluvput API parameters:

hdl 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.

Usage notes:

This API is used by the backup utility.

Return codes:

Table 87. 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 (see note below).	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT (see note below).	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT (see note below).	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT (see note below).	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.

sqluvput - Write data to a vendor device

Table 87. Valid Return Codes for sqluvput and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT (see note below).	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 (see note below).	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT (see note below).	The session will be terminated.

Note: Next call: 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.

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671
- “Data ” on page 704
- “Return_code ” on page 705

sqluvend - Unlink a vendor device and release its resources

Unlinks a vendor device and frees all of its related resources. All unused resources (for example, allocated space and file handles) must be released before the sqluvend API call returns to DB2.

Authorization:

None

Required connection:

Database

API include file:

sqluvend.h

API and data structure syntax:

```
int sqluvend ( sqlint32      action,
              void *hdlle,
              struct Init_output *in_out,
              struct Return_code *return_code);
```

sqluvend API parameters:

action Input. Used to commit or abort the session:

- SQLUV_COMMIT (0 = to commit)
- SQLUV_ABORT (1 = to abort)

hdlle Input. Pointer to the Init_output structure.

in_out 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 API 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.

sqluvend - Unlink a vendor device and release its resources

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 88. 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	

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671
- “Init_output ” on page 703
- “Return_code ” on page 705

sqluvdel - Delete committed session

Deletes committed sessions from a vendor device.

Authorization:

None

Required connection:

Database

API include file:

sqluvend.h

API and data structure syntax:

```
int sqluvdel ( struct Init_input *in,
              struct Init_output *vendorDevData,
              struct Return_code *return_code);
```

sqluvdel API parameters:

in Input. Space allocated for Init_input and Return_code.

vendorDevData

Output. Structure containing the output returned by the vendor device.

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 API 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 the sqluvdel API, an initial fatal error must have occurred.

Return codes:

Table 89. Valid return codes for sqluvdel and resulting database server action

Literal in header file	Description	Probable next call
SQLUV_OK	Operation successful	No further calls
SQLUV_DELETE_FAILED	Delete request failed	No further calls

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671
- “Init_input ” on page 701
- “Init_output ” on page 703

sqluvdel - Delete committed session

- "Return_code " on page 705

db2VendorQueryApiVersion - Get the supported level of the vendor storage API

db2VendorQueryApiVersion - Get the supported level of the vendor storage API

Determines which level of the vendor storage API is supported by the backup and restore vendor storage plug-in. If the specified vendor storage plug-in is not compatible with DB2, then it will not be used.

If a vendor storage plug-in does not have this API implemented for logs, then it cannot be used and DB2 will report an error. This will not affect images that currently work with existing vendor libraries.

Authorization:

None

Required connection:

Database.

API include file:

db2VendorApi.h

API and data structure syntax:

```
void db2VendorQueryApiVersion ( db2UInt32 * supportedVersion );
```

db2VendorQueryApiVersion API parameters:

supportedVersion

Output. Returns the version of the vendor storage API the vendor library supports.

Usage notes:

This API will be called before any other vendor storage APIs are invoked.

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

db2VendorGetNextObj - Get next object on device

This API is called after a query has been set up (using the sqluvint API) to get the next object (image or archived log file) that matches the search criteria. Only one search for either images or log files can be set up at one time.

Authorization:

None

Required connection:

Database.

API include file:

db2VendorApi.h

API and data structure syntax:

```
int db2VendorGetNextObj ( void                * vendorCB,  
                        struct db2VendorQueryInfo * queryInfo,  
                        struct Return_code      * returnCode);
```

```
typedef struct db2VendorQueryInfo  
{  
    db2UInt64 sizeEstimate;  
    db2UInt32 type;  
    SQL_PDB_NODE_TYPE dbPartitionNum;  
    db2UInt16 sequenceNum;  
    char db2Instance[SQL_INSTNAME_SZ + 1];  
    char dbname[SQL_DBNAME_SZ + 1];  
    char dbalias[SQL_ALIAS_SZ + 1];  
    char timestamp[SQLU_TIME_STAMP_LEN + 1];  
    char filename[DB2VENDOR_MAX_FILENAME_SZ + 1];  
    char owner[DB2VENDOR_MAX_OWNER_SZ + 1];  
    char mgmtClass[DB2VENDOR_MAX_MGMTCLASS_SZ + 1];  
    char oldestLogfile[DB2_LOGFILE_NAME_LEN + 1];  
} db2VendorQueryInfo;
```

db2VendorGetNextObj API parameters:

vendorCB

Input. Pointer to space allocated by the vendor library.

queryInfo

Output. Pointer to a db2VendorQueryInfo structure to be filled in by the vendor library.

returnCode

Output. The return code from the API call.

db2VendorQueryInfo data structure parameters:

sizeEstimate

Specifies the estimated size of the object.

type Specifies the image type if the object is a backup image.

dbPartitionNum

Specifies the number of the database partition that the object belongs to.

db2VendorGetNextObj - Get next object on device

sequenceNum

Specifies the file extension for the backup image. Valid only if the object is a backup.

db2Instance

Specifies the name of the instance that the object belongs to.

dbname

Specifies the name of the database that the object belongs to.

dbalias

Specifies the alias of the database that the object belongs to.

timestamp

Specifies the time stamp used to identify the backup image. Valid only if the object is a backup image.

filename

Specifies the name of the object if the object is a load copy image or an archived log file.

owner Specifies the owner of the object.

mgmtClass

Specifies the management class the object was stored under (used by TSM).

oldestLogfile

Specifies the oldest log file stored with a backup image.

Usage notes:

Not all parameters will pertain to each object or each vendor. The mandatory parameters that need to be filled out are db2Instance, dbname, dbalias, timestamp (for images), filename (for logs and load copy images), owner, sequenceNum (for images) and dbPartitionNum. The remaining parameters will be left for the specific vendors to define. If a parameter does not pertain, then it should be initialized to "" for strings and 0 for numeric types.

Related reference:

- "DB2 APIs for backup and restore to storage managers" on page 671
- "Return_code " on page 705

DB2_info

Contains information about the DB2 product and the database that is being backed up or restored. This structure is used to identify DB2 to the vendor device and to describe a particular session between DB2 and the vendor device. It is passed to the backup and restore vendor storage plug-in as part of the Init_input data structure.

Table 90. Fields in the DB2_info Structure.

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 90. Fields in the DB2_info Structure. (continued)

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.

Table 90. Fields in the DB2_info Structure. (continued)

Field Name	Data Type	Description
nodeNum	SQL_PDB_NODE_TYPE	Node number. Numbers greater than 255 are supported by the vendor interface.

Note: All char data type fields are null-terminated strings.

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.

API and data structure syntax:

```
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;
    sqlint32 max_bytes_per_txn;
    char *image_filename;
    void *reserve;
    char *nodename;
    char *password;
    char *owner;
    char *mcNameP;
    SQL_PDB_NODE_TYPE nodeNum;
} DB2_info ;
```

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

Vendor_info

Contains information, returned to DB2 as part of the Init_output structure, identifying the vendor and the version of the vendor device.

Table 91. Fields in the Vendor_info Structure.

Field Name	Data Type	Description
vendor_id	char	An identifier for the vendor. Maximum length of the string it points to is 64 characters.
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 API 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.

Note: All char data type fields are NULL-terminated strings.

API and data structure syntax:

Vendor_info

```
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;
```

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

Init_input

Contains information provided by DB2 to set up and to establish a logical link with a vendor device. This data structure is used by DB2 to send information to the backup and restore vendor storage plug-in through the sqluvint and sqluvdel APIs.

Table 92. Fields in the Init_input Structure.

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. This field is a NULL-terminated string.
num_sessions	unsigned short	Number of sessions requested by the user when a backup or a restore operation was invoked.

API and data structure syntax:

Init_input

```
typedef struct Init_input
{
    struct DB2_info *DB2_session;
    unsigned short  size_options;
    sqluint32 size_HI_order;
    sqluint32 size_LOW_order;
    void *options;
    void *reserve;
    char *prompt_lvl;
    unsigned short  num_sessions;
} Init_input;
```

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

Init_output

Contains a control block for the session and information returned by the backup and restore vendor storage plug-in to DB2. This data structure is used by the sqluvint and sqluvdel APIs.

Table 93. 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.

API and data structure syntax:

```
typedef struct Init_output
{
    struct Vendor_info * vendor_session;
    void                * pVendorCB;
    void                * reserve;
} Init_output ;
```

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

Data

Contains data transferred between DB2 and a vendor device. This structure is used by the `sqlvput` API when data is being written to the vendor device and by the `sqlvget` API when data is being read from the vendor device.

Table 94. Fields in the Data Structure

Field Name	Data Type	Description
<code>obj_num</code>	<code>sqlint32</code>	The sequence number assigned by DB2 during a backup operation.
<code>buff_size</code>	<code>sqlint32</code>	The size of the buffer.
<code>actual_buf_size</code>	<code>sqlint32</code>	The actual number of bytes sent or received. This must not exceed <code>buff_size</code> .
<code>dataptr</code>	<code>void</code>	Pointer to the data buffer. DB2 allocates space for the buffer.
<code>reserve</code>	<code>void</code>	Reserved for future use.

API and data structure syntax:

```
typedef struct Data
{
    sqlint32 obj_num;
    sqlint32 buff_size;
    sqlint32 actual_buff_size;
    void *dataptr;
    void *reserve;
} Data;
```

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

Return_code

Contains the return code for and a short explanation of the error being returned to DB2 by the backup and restore vendor storage plug-in. This data structure is used by all the vendor storage plug-in APIs.

Table 95. Fields in the Return_code Structure

Field Name	Data Type	Description
return_code (see note below)	sqlint32	Return code from the vendor API.
description	char	A short description of the return code.
reserve	void	Reserved for future use.

Note: 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.

API and data structure syntax:

```
typedef struct Return_code
{
    sqlint32 return_code;
    char description[SQLUV_COMMENT_LEN];
    void *reserve;
} Return_code;
```

Related reference:

- “DB2 APIs for backup and restore to storage managers” on page 671

Compression plug-ins for backup and restore

DB2 APIs for using compression with backup and restore operations

DB2 provides APIs that can be used by third-party compression products to compress and decompress backup images. This interface is designed to augment or replace the compression library that is supported as a standard part of DB2. The compression plug-in interface can be used with the backup and restore DB2 APIs or the backup and restore plug-ins for vendor storage devices.

DB2 defines a set of API prototypes that provide a general purpose interface for compression and decompression that can be used by many vendors. These APIs are to be provided by the vendor in a shared library on Linux and UNIX systems, or DLL on the Windows operating system. When the APIs are invoked by DB2, the shared library or DLL specified by the calling backup or restore routine is loaded and the APIs provided by the vendor are called to perform the required tasks.

Operational overview

Eight APIs are defined to interface DB2 and the vendor product:

- InitCompression - Initialize the compression library
- GetSavedBlock - Get vendor block for backup image
- Compress - Compress a block of data

DB2 APIs for using compression with backup and restore operations

- GetMaxCompressedSize - Estimate largest possible buffer size
- TermCompression - Terminate the compression library
- InitDecompression - Initialize the decompression library
- Decompress - Decompress a block of data
- TermDecompression - Terminate the decompression library

DB2 will provide the definition for the COMPR_DB2INFO structure; the vendor will provide definitions for each of the other structures and APIs for using compression with backup and restore. The structures, prototypes, and constants are defined in the file `sqlucompr.h`, which is shipped with DB2.

DB2 will call these APIs, and they should be provided by the vendor product in a shared library on Linux and UNIX 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 might compromise data integrity of the database.

Sample calling sequence

For backup, the following sequence of calls is issued by DB2 for each session:

InitCompression

followed by 0 to 1

GetMaxCompressedSize
Compress

followed by 1

TermCompress

For restore, the sequence of calls for each session is:

InitDecompression

followed by 1 to n

Decompress

followed by 1

TermCompression

Compression plug-in interface return codes

The following are the return codes that the APIs might return. Except where specified, DB2 will terminate the backup or restore when any non-zero return code is returned.

SQLUV_OK

0

Operation succeeded

SQLUV_BUFFER_TOO_SMALL

100

DB2 APIs for using compression with backup and restore operations

Target buffer is too small. When indicated on backup, the `tgtAct` field shall indicate the estimated size required to compress the object. DB2 will retry the operation with a buffer at least as large as specified. When indicated on restore, the operation will fail.

`SQLUV_PARTIAL_BUFFER`

101

A buffer was partially compressed. When indicated on backup, the `srcAct` field shall indicate the actual amount of data actually compressed and the `tgtAct` field shall indicate the actual size of the compressed data. When indicated on restore, the operation will fail.

`SQLUV_NO_MEMORY`

102

Out of memory

`SQLUV_EXCEPTION`

103

A signal or exception was raised in the code.

`SQLUV_INTERNAL_ERROR`

104

An internal error was detected.

The difference between `SQLUV_BUFFER_TOO_SMALL` and `SQLUV_PARTIAL_BUFFER` is that when `SQLUV_PARTIAL_BUFFER` is returned, DB2 will consider the data in the output buffer to be valid.

Related concepts:

- “DB2 database system plug-ins for customizing database management” on page 591

Related reference:

- “Compress - Compress a block of data” on page 710
- “`COMPR_CB`” on page 721
- “`COMPR_DB2INFO`” on page 717
- “`COMPR_PIINFO`” on page 719
- “`db2Backup` - Back up a database or table space” on page 41
- “`db2Restore` - Restore a database or table space” on page 284
- “Decompress - Decompress a block of data” on page 715
- “`GetMaxCompressedSize` - Estimate largest possible buffer size” on page 712
- “`GetSavedBlock` - Get the vendor of block data for the backup image” on page 709
- “`InitCompression` - Initialize the compression library” on page 708
- “`TermDecompression` - Stop the decompression library” on page 716
- “`InitDecompression` - Initialize the decompression library” on page 714
- “`TermCompression` - Stop the compression library” on page 713

InitCompression - Initialize the compression library

Initializes the compression library. DB2 will pass the db2Info and piInfo structures. The library will fill in the appropriate parameters of piInfo, and will allocate pCB and return a pointer to the allocated memory.

Authorization:

None

Required connection:

None

API include file:

sqlucompr.h

API and data structure syntax:

```
int InitCompression(  
    const struct COMPR_DB2INFO  
        *db2Info,  
    struct COMPR_PIINFO  
        *piInfo,  
    struct COMPR_CB **pCB);
```

InitCompression API parameters:

db2Info

Input. Describes the database being backed up and gives details about the DB2 environment where the operation is occurring.

piInfo Output. This structure is used by the plug-in library to describe itself to DB2. It is allocated and initialized by DB2 and the key parameters are filled in by the plug-in library.

pCB Output. This is the control block used by the compression library. The plug-in library is responsible for memory management of the structure.

Related reference:

- "DB2 APIs for using compression with backup and restore operations" on page 705
- "COMPR_CB " on page 721
- "COMPR_DB2INFO " on page 717
- "COMPR_PIINFO " on page 719

GetSavedBlock - Get the vendor of block data for the backup image

Gets the vendor-specific block of data to be saved with the backup image. If the library returned a non-zero value for piInfo->savedBlockSize, DB2 will call GetSavedBlock using that value as blockSize. The plug-in library writes data of the given size to the memory referenced by data. This API will be called during initial data processing by the first db2bm process for backup only. Even if parallelism > 1 is specified in the db2Backup API, this API will be called only once per backup.

Authorization:

None

Required connection:

None

API include file:

sqlucompr.h

API and data structure syntax:

```
int GetSavedBlock(  
    struct COMPR_CB *pCB,  
    db2Uint32 blockSize,  
    void *data);
```

GetSavedBlock API parameters:

pCB Input. This is the control block that was returned by the InitCompression API call.

blocksize

Input. This is the size of the block that was returned in piInfo->savedBlockSize by the InitCompression API call.

data Output. This is the vendor-specific block of data to be saved with the backup image.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705
- “COMPR_CB ” on page 721

Compress - Compress a block of data

Compress a block of data. The `src` parameter points to a block of data that is `srcLen` bytes in size. The `tgt` parameter points to a buffer that is `tgtSize` bytes in size. The plug-in library compresses the data at address `src` and writes the compressed data to the buffer at address `tgt`. The actual amount of uncompressed data that was compressed is stored in `srcAct`. The actual size of the compressed data is returned as `tgtAct`.

Authorization:

None

Required connection:

None

API include file:

`sqlucompr.h`

API and data structure syntax:

```
int Compress(  
    struct COMPR_CB *pCB,  
    const char *src,  
    db2int32 srcSize,  
    db2UInt32 srcGran,  
    char *tgt,  
    db2int32 tgtSize,  
    db2int32 *srcAct,  
    db2int32 *tgtAct,  
    db2UInt32 *tgtCRC);
```

Compress API parameters:

pCB Input. This is the control block that was returned by the `InitCompression` API call.

src Input. Pointer to the block of data to be compressed.

srcLen Input. Size in bytes of the block of data to be compressed.

srcGran

Input. If the library returned a value of 1 for `piInfo->useGran`, `srcGran` specifies the \log_2 of the page size of the data. (For example, if the page size of the data is 4096 bytes, `srcGran` is 12.) The library ensures that the amount of data actually compressed (`srcAct`) is an exact multiple of this page size. If the library sets the `useGran` flag, DB2 is able to use a more efficient algorithm for fitting the compressed data into the backup image. This means that both the performance of the plug-in will be better and that the compressed backup image will be smaller. If the library returned a value of 0 for `piInfo->srcGran`, the granularity is 1 byte.

tgt Input and output. Target buffer for compressed data. DB2 will supply this target buffer and the plug-in will compress the data at `src` and write compressed data here.

tgtSize

Input. Size in bytes of the target buffer.

Compress - Compress a block of data

srcAct Output. Actual amount in bytes of uncompressed data from src that was compressed.

tgtAct Output. Actual amount in bytes of compressed data stored in tgt.

tgtCRC

Output. If the library returned a value of 1 for piInfo->useCRC, the CRC value of the uncompressed block is returned as tgtCRC. If the library returned a value of 0 for piInfo->useCRC, tgtCRC will be a null pointer.

Related reference:

- "DB2 APIs for using compression with backup and restore operations" on page 705
- "COMPR_CB " on page 721

GetMaxCompressedSize - Estimate largest possible buffer size

Estimates the size of the largest possible buffer required to compress a block of data. `srcLen` indicates the size of a block of data about to be compressed. The library returns the theoretical maximum size of the buffer after compression as `tgtLen`.

DB2 will use the value returned as `tgtLen` to optimize its use of memory internally. The penalty for not calculating a value or for calculating an incorrect value is that DB2 will have to call the Compress API more than once for a single block of data, or that it may waste memory from the utility heap. DB2 will still create the backup correctly, regardless of the values returned.

Authorization:

None

Required connection:

None

API include file:

`sqlucompr.h`

API and data structure syntax:

```
int GetMaxCompressedSize(  
    struct COMPR_CB *pCB,  
    db2Uint32 srcLen);
```

GetMaxCompressedSize API parameters:

pCB Input. This is the control block that was returned by the `InitCompression` API call.

srcLen Input. Size in bytes of a block of data about to be compressed.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705
- “COMPR_CB ” on page 721

TermCompression - Stop the compression library

Terminates the compression library. The library will free the memory used for pCB.

Authorization:

None

Required connection:

None

API include file:

sqlucompr.h

API and data structure syntax:

```
int TermCompression(  
    struct COMPR_CB *pCB);
```

TermCompression API parameters:

pCB Input. This is the control block that was returned by the InitCompression API call.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705
- “COMPR_CB ” on page 721

InitDecompression - Initialize the decompression library

Initializes the decompression library. DB2 will pass the db2Info structure. The library will allocate pCB and return a pointer to the allocated memory.

Authorization:

None

Required connection:

None

API include file:

sqlucompr.h

API and data structure syntax:

```
int InitDecompression(  
    const struct COMPR_DB2INFO  
        *db2Info,  
    struct COMPR_CB **pCB);
```

InitDecompression API parameters:

db2Info

Input. Describes the database being backed up and gives details about the DB2 environment where the operation is occurring.

pCB Output. This is the control block used by the decompression library. The plug-in library is responsible for memory management of the structure.

Related reference:

- "DB2 APIs for using compression with backup and restore operations" on page 705
- "COMPR_CB " on page 721
- "COMPR_DB2INFO " on page 717

Decompress - Decompress a block of data

Decompresses a block of data. The `src` parameter points to a block of data that is `srcLen` bytes in size. The `tgt` parameter points to a buffer that is `tgtSize` bytes in size. The plug-in library decompresses the data at address `src` and writes the uncompressed data to the buffer at address `tgt`. The actual size of the uncompressed data is returned as `tgtLen`. If the library returned a value of 1 for `piInfo->useCRC`, the CRC of the uncompressed block is returned as `tgtCRC`. If the library returned a value of 0 for `piInfo->useCRC`, `tgtLen` will be a null pointer.

Authorization:

None

Required connection:

None

API include file:

`sqlucompr.h`

API and data structure syntax:

```
int Decompress(
    struct COMPR_CB *pCB,
    const char *src,
    db2int32 srcSize,
    char *tgt,
    db2int32 tgtSize,
    db2int32 *tgtLen,
    db2Uint32 *tgtCRC);
```

Decompress API parameters:

pCB Input. This is the control block that was returned by the `InitDecompression` API call.

src Input. Pointer to the block of data to be decompressed.

srcLen Input. Size in bytes of the block of data to be decompressed.

tgt Input and output. Target buffer for decompressed data. DB2 will supply this target buffer and the plug-in will decompress the data at `src` and write decompressed data here.

tgtSize Input. Size in bytes of the target buffer.

tgtLen Output. Actual amount in bytes of decompressed data stored in `tgt`.

tgtCRC Output. If the library returned a value of 1 for `piInfo->useCRC`, the CRC value of the uncompressed block is returned as `tgtCRC`. If the library returned a value of 0 for `piInfo->useCRC`, `tgtCRC` will be a null pointer.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705
- “COMPR_CB ” on page 721

TermDecompression - Stop the decompression library

Terminates the decompression library. The library will free the memory used for pCB. All the memory used internally by the compression APIs will be managed by the library. The plug-in library will also manage memory used by the COMPR_CB structure. DB2 will manage the memory used for the data buffers (the src and tgt parameters in the compression APIs).

Authorization:

None

Required connection:

None

API include file:

sqlucompr.h

API and data structure syntax:

```
int TermDecompression(  
    struct COMPR_CB *pCB);
```

TermDecompression API parameters:

pCB Input. This is the control block that was returned by the InitDecompression API call.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705
- “COMPR_CB ” on page 721

COMPR_DB2INFO

Describes the DB2 environment. DB2 allocates and defines this structure and passes it in as a parameter to the InitCompression and InitDecompression APIs. This structure describes the database being backed up or restored and gives details about the DB2 environment where the operation is occurring. The dbalias, instance, node, catnode, and timestamp parameters are used to name the backup image.

API and data structure syntax:

```
struct COMPR_DB2INFO {
    char tag[16];
    db2UInt32 version;
    db2UInt32 size;
    char dbalias[SQLU_ALIAS_SZ+1];
    char instance[SQL_INSTNAME_SZ+1];
    SQL_PDB_NODE_TYPE node;
    SQL_PDB_NODE_TYPE catnode;
    char timestamp[SQLU_TIME_STAMP_LEN+1];
    db2UInt32 bufferSize;
    db2UInt32 options;
    db2UInt32 bkOptions;
    db2UInt32 db2Version;
    db2UInt32 platform;
    db2int32 comprOptionsByteOrder;
    db2UInt32 comprOptionsSize;
    void *comprOptions;
    db2UInt32 savedBlockSize;
    void *savedBlock;
};
```

COMPR_DB2INFO data structure parameters:

tag Used as an eye catcher for the structure. This is always set to the string "COMPR_DB2INFO \0".

version

Indicates which version of the structure is being used so APIs can indicate the presence of additional fields. Currently, the version is 1. In the future there may be more parameters added to this structure.

size Specifies the size of the COMPR_DB2INFO structure in bytes.

dbalias

Database alias. For restore operations, dbalias refers to the alias of the source database.

instance

Instance name.

node Node number.

catnode

Catalog node number.

timestamp

The timestamp of the image being backed up or restored.

bufferSize

Specifies the size of a transfer buffer (in 4K pages).

options

The iOptions parameter specified in the db2Backup API or the db2Restore API.

bkOptions

For restore operations, specifies the iOptions parameter that was used in the db2Backup API when the backup was created. For backup operations, it is set to zero.

db2Version

Specifies the version of the DB2 engine.

platform

Specifies the platform on which the DB2 engine is running. The value will be one of the ones listed in sqlmon.h (located in the include directory).

comprOptionsByteOrder

Specifies the byte-order used on the client where the API is being run. DB2 will do no interpretation or conversion of the data passed through as comprOptions, so this field should be used to determine whether the data needs to be byte reversed before being used. Any conversion must be done by the plug-in library itself.

comprOptionsSize

Specifies the value of the piComprOptionsSize parameter in the db2Backup and db2Restore APIs.

comprOptions

Specifies the value of the piComprOptions parameter in the db2Backup and db2Restore APIs.

savedBlockSize

Size in bytes of savedBlock.

savedBlock

DB2 allows the plug-in library to save an arbitrary block of data in the backup image. If such a block of data was saved with a particular backup, it will be returned in these fields on the restore operation. For backup operations, these fields are set to zero.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705

COMPR_PIINFO

This structure is used by the plug-in library to describe itself to DB2. This structure is allocated and initialized by DB2, and the key fields are filled in by the plug-in library on the InitCompression API call.

API and data structure syntax:

```
struct COMPR_PIINFO {
    char tag[16];
    db2Uint32 version;
    db2Uint32 size;
    db2Uint32 useCRC;
    db2Uint32 useGran;
    db2Uint32 useAllBlocks;
    db2Uint32 savedBlockSize;
};
```

COMPR_PIINFO data structure parameters:

tag Used as an eye catcher for the structure. (It is set by DB2.) This is always set to the string "COMPR_PIINFO \0".

version

Indicates which version of the structure is being used so APIs can indicate the presence of additional fields. Currently, the version is 1.

(It is set by DB2.) In the future there may be more fields added to this structure.

size Indicates the size of the COMPR_PIINFO structure (in bytes). (It is set by DB2.)

useCRC

DB2 allows compression plug-ins to use a 32-bit CRC or checksum value to validate the integrity of the data being compressed and decompressed.

If the library uses such a check, it will set this field to 1. Otherwise, it will set the field to 0.

useGran

If the compression routine is able to compress data in arbitrarily-sized increments, the library will set this field to 1. If the compression routine compresses data only in byte-sized increments, the library will set this field to 0. See the description of the srcGran parameter of Compress API for details of the implications of setting this indicator.

For restore operations, this parameter is ignored.

useAllBlocks

Specifies whether DB2 should back up a compressed block of data that is larger than the original uncompressed block. By default, DB2 will store data uncompressed if the compressed version is larger, but under some circumstances the plug-in library will wish to have the compressed data backed up anyway. If DB2 is to save the compressed version of the data for all blocks, the library will set this value to 1. If DB2 is to save the compressed version of the data only when it is smaller than the original data, the library will set this value to 0. For restore operations, this field is ignored.

savedBlockSize

DB2 allows the plug-in library to save an arbitrary block of data in the backup image. If such a block of data is to be saved with a particular

COMPR_PIINFO

backup, the library will set this parameter to the size of the block to be allocated for this data. (The actual data will be passed to DB2 on a subsequent API call.) If no data is to be saved, the plug-in library will set this parameter to zero. For restore operations, this parameter is ignored.

Related reference:

- “DB2 APIs for using compression with backup and restore operations” on page 705

COMPR_CB

This structure is used internally by the plug-in library as the control block. It contains data used internally by compression and decompression APIs. DB2 passes this structure to each call it makes to the plug-in library, but all aspects of the structure are left up to the library, including the definition of the structure's parameters and memory management of the structure.

API and data structure syntax:

```
struct COMPR_CB;
```

Related reference:

- "DB2 APIs for using compression with backup and restore operations" on page 705

Appendix A. Precompiler customization APIs

A set of documented APIs to enable other application development tools to implement precompiler support for DB2 directly within their products. For example, IBM COBOL on AIX uses this interface. Information on the set of Precompiler Services APIs is available from the PDF file, `prepapi.pdf`, at the DB2 application development Web site:

<http://www.ibm.com/software/data/db2/udb/ad>

Related reference:

- “Contacting IBM” on page 779

Precompiler customization APIs

Appendix B. DB2 log records

This section describes the structure of the DB2 log records returned by the `db2ReadLog` API when the `iFilterOption` input value `DB2READLOG_FILTER_ON` is specified. Only propagated log records are returned when this value is used. Only propagated log records are documented. All other log records are intended for IBM internal use only and are therefore not documented.

All DB2 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 727.

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 96. DB2 Log Records

Data Manager	“Initialize Table” on page 730	New permanent table creation.
Data Manager	“Import Replace (Truncate)” on page 732	Import replace activity.
Data Manager	“Rollback Insert” on page 733	Rollback row insert.
Data Manager	“Reorg Table” on page 733	REORG committed.
Data Manager	“Create index, drop index log records” on page 733	Index activity.
Data Manager	“Create table, drop table, rollback create table, rollback drop table log records” on page 734	Table activity.
Data Manager	“Alter Table Attribute” on page 734	Propagation, check pending, and append mode activity.
Data Manager	“Alter Table Add Columns, Rollback Add Columns” on page 735	Adding columns to existing tables.
Data Manager	“Insert Record, Rollback Delete Record, Rollback Update Record” on page 736	Table record activity.

Table 96. DB2 Log Records (continued)

Data Manager	“Update Record” on page 740	Row updates where storage location not changed.
Long Field Manager	“Add/Delete/Non-update Long Field Record” on page 742	Long field record activity.
Transaction Manager	“Normal commit” on page 743	Transaction commits.
Transaction Manager	“Heuristic Commit” on page 744	Indoubt transaction commits.
Transaction Manager	“MPP Coordinator Commit” on page 744	Transaction commits. This is written on a coordinator node for an application that performs updates on at least one subordinator node.
Transaction Manager	“MPP Subordinator Commit” on page 744	Transaction commits. This is written on a subordinator node.
Transaction Manager	“Normal Abort” on page 745	Transaction aborts.
Transaction Manager	“Heuristic Abort” on page 745	Indoubt transaction aborts.
Transaction Manager	“Local Pending List” on page 746	Transaction commits with a pending list existing.
Transaction Manager	“Global Pending List” on page 746	Transaction commits (two-phase) with a pending list existing.
Transaction Manager	“XA Prepare” on page 746	XA transaction preparation in two-phase commit environments.
Transaction Manager	“MPP Subordinator Prepare” on page 747	MPP transaction preparation in two-phase commit environments. This log record only exists on subordinator nodes.
Transaction Manager	“Backout free” on page 748	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 749	Catalog migration starts.
Utility Manager	“Migration End” on page 749	Catalog migration completes.
Utility Manager	“Load Start” on page 749	Table load starts.
Utility Manager	“Backup End” on page 750	Backup activity completes.
Utility Manager	“Table Space Rolled Forward” on page 750	Table space rollforward completes.
Utility Manager	“Table Space Roll Forward to Point in Time Starts” on page 750	Marks the beginning of a table space rollforward to a point in time.

Table 96. DB2 Log Records (continued)

Utility Manager	“Table Space Roll Forward to Point in Time Ends” on page 751	Marks the end of a table space rollforward to a point in time.
-----------------	--	--

Log Manager Header

All DB2 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 97. 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 96 on page 725.)	short	4(2)
Log record general flag ¹	short	6(2)
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)
<i>Total Length for Log Manager Log Record Header:</i> <ul style="list-style-type: none"> • Non Compensation: 20 bytes • Compensation: 26 bytes • Propagatable Compensation: 32 bytes 		

Notes:

1. Log record general flag constants

Redo Always	0x0001
Propagatable	0x0002
Temp Table	0x0004
Tablespace rollforward undo	0x0008
Singular transaction (no commit/rollback)	0x0010
Conditionally Recoverable	0x0080
Tablespace rollforward at check constraint process	0x0100

2. Log Sequence Number (LSN)

Log Manager Header

A unique log record identifier representing the relative byte address of the log record within the database log.

```
SQLU_LSN: union { unsigned char [6] ;
                  unsigned short [3] ;
                }
```

3. Transaction Identifier (TID)

A unique log record identifier representing the transaction.

```
SQLU_TID: union { unsigned char [6] ;
                  unsigned short [3] ;
                }
```

4. Record ID (RID)

A unique number identifying the position of a record.

```
RID: Page number char [4];
     slot number char [2];
```

Table 98. Log Manager Log Record Header Log Type Values and Definitions

Value	Definition
0x0041	Normal abort
0x0042	Backout free
0x0043	Compensation
0x0049	Heuristic abort
0x004A	Load start
0x004E	Normal log record
0x004F	Backup end
0x0051	Global pending list
0x0052	Redo
0x0055	Undo
0x0056	Migration begin
0x0057	Migration end
0x0061	Datalink manager log record
0x0069	Information only
0x006F	Backup start
0x0071	Table Space Roll Forward to Point in Time Ends
0x0081	MPP prepare
0x0082	XA prepare
0x0083	TM ^a prepare
0x0084	Normal commit
0x0085	MPP subordinate commit
0x0086	MPP coordinator commit
0x0087	Heuristic commit
0x0089	Table Space Roll Forward to Point in Time Starts
0x008A	Local pending list

¹ TM = Transaction Manager

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 99. DMS Log Record Header Structure (DMSLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=1)	unsigned char	0(1)
Function identifier (See Table 98 on page 728.)	unsigned char	1(1)
Table identifiers		
Table space identifier	unsigned short	2(2)
Table identifier	unsigned short	4(2)
<i>Total Length: 6 bytes</i>		

Table 100. DMS Log Record Header Structure Function Identifier Values and Definitions

Value	Definition
102	Add columns to table
104	Undo add columns
110	Undo insert record
111	Undo delete record
112	Undo update record
113	Alter column length
115	Undo alter column length
122	Rename a schema or table
123	Undo rename a schema or table
124	Alter table attribute
128	Initialize table
131	Undo insert record to empty page
134	Insert record group
135	Undo insert record group
161	Delete record
162	Insert record
163	Update record
164	Delete record to empty page
165	Insert record to empty page
166	Undo delete record to empty page
167	Insert multiple records
168	Undo insert multiple records

Data Manager Log Records

Table 101. DOM Log Record Header Structure (DOMLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=4)	unsigned char	0(1)
Function identifier (See Table 100 on page 729.)	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)
<i>Total Length: 12 bytes</i>		

Table 102. DOM Log Record Header Structure Function Identifier Values and Definitions

Value	Definition
2	Create index
3	Drop index
4	Drop table
5	Undo drop table
11	Truncate table (import replace)
12	Activate NOT LOGGED INITIALLY
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 creates the DATA and Block Map storage objects, and before any log records

that create the LF and LOB storage objects. This is a Redo log record.

Table 103. Initialize Table Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
File create LSN	SQLU_LSN	6(6)
Internal	variable	12(72)
Table description length	squint32	84(4)
Table description record	variable	88(variable)
<i>Total Length: 88 bytes plus table description record length</i>		

Table 104. Table description record

Description	Type	Offset (Bytes)
record type	unsigned char	0(1)
reserved	char	1(1)
number of columns	unsigned short	2(2)
array of column descriptor	variable long	4(variable)
<i>Total length: 4 bytes plus the array of column descriptor length</i>		

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, generated, or compress type default

Data Manager Log Records

ISNULL	0x0001
NONULLS	0x0002
TYPE_DEFAULT	0x0004
USER_DEFAULT	0x0008
GENERATED	0x0040
COMPRESS_SYSTEM_DEFAULT	0x0080

- field offset (unsigned short, 2 bytes) This is the offset from the start of the fixed-length portion of user record to where the field's fixed value can be found.

Table Description Record: LOB column 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.

If there are datalinks columns, there will be another array of information for the datalinks columns.

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 table identifiers in the log header identify the table being truncated (IMPORT REPLACE). This is a normal log record.

Table 105. Import Replace (Truncate) Log Record Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
internal	variable	12(variable)
<i>Total Length: 12 bytes plus variable length</i>		

Activate Not Logged Initially

The activate not logged initially log record is written when a user issues an ALTER TABLE statement that includes the ACTIVATE NOT LOGGED INITIALLY clause. This is a normal log record.

Table 106. Active Not Logged Initially Log Record Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
internal	variable	12(variable)
Long Tablespace ID*	unsigned short	16(2)
Index Tablespace ID*	unsigned short	18(2)
Index Object ID	unsigned short	20(2)
LF Object ID	unsigned short	22(2)

Table 106. Active Not Logged Initially Log Record Structure (continued)

Description	Type	Offset (Bytes)
LOB Object ID	unsigned short	24(2)
<i>Total Length: 26 bytes</i>		

* Same as Table Space Identifiers in the DOM header; it is a unique identifier for each table space defined in the database.

The activate not logged initially log record is written when a user issues an ALTER TABLE statement that includes the ACTIVATE NOT LOGGED INITIALLY clause. This is a normal log record.

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 107. Rollback Insert Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Record Length	unsigned short	6(2)
RID	char[]	8(46)
Free space	unsigned short	14(2)
<i>Total Length: 16 bytes</i>		

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 108. Reorg Table Log Record Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(392)
Index token ¹	unsigned short	404(2)
Temporary table space ID ²	unsigned short	406(2)
<i>Total Length: 408 bytes</i>		

Notes:

1. If the value of the index token is not 0, it is the index by which the reorg is clustered (clustering index).
2. If the value of the temporary table space ID is not 0, it is the system temporary table space that was used to build the reorganized table.

Create index, drop index log records

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

Data Manager Log Records

- 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 109. Create Index, Drop Index Log Records Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Padding	char[]	12(2)
Index token	unsigned short	14(2)
Index root page	sqluint32	16(4)
<i>Total Length: 20 bytes</i>		

Create table, drop table, rollback create table, rollback drop table log records

These log records are written when the DATA object for a permanent table is created or dropped. For creation of an MDC table, there is also a create table log record for creation of the Block Map object. The DATA object (and block Map object if applicable) is created during a CREATE TABLE operation, 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 110. 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)
<i>Total Length: 84 bytes</i>		

Alter Table Attribute

The alter table attribute log record is written when the state of a table is changed using the ALTER TABLE statement or as a result of adding or validating constraints. This can be a Normal or Compensation log record.

Table 111. Alter Table Attribute, Undo Alter Table Attribute

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	unsigned short	6(2)
Alter bit (attribute) mask	sqluint64	8(8)
Alter bit (attribute) values	sqluint64	16(8)
<i>Total Length: 16 bytes</i>		

Attribute Bits:

```

0x00000001 Propagation
0x00000002 Check Pending
0x00000010 Value Compression
0x00010000 Append Mode
0x00200000 LF Propagation

```

All other bits are for internal use.

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:
 - One 8-byte element for each column.
 - If there are any LOB columns, one 12 byte element for each LOB column. This follows the array of 8 byte elements.
 - If there are datalink columns, one 28 byte element for each datalink column. This follows the array of 12 bytes elements.

Alter table add columns is a Normal log record. Rollback add columns is a Compensation log record.

Table 112. Alter Table Add Columns, Rollback Add Columns Log Records Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordheader	0(6)
Padding	unsigned short	6(2)
Old column count	sqluint32	8(4)
New column count	sqluint32	12(4)
Old parallel arrays ¹	variable	16(variable)
New parallel arrays	variable	variable
<i>Total Length: 16 bytes plus 2 sets of parallel arrays.</i>		

Array Elements:

1. The lengths of the elements in this array are defined as follows:
 - If the element is a column descriptor, the element length is 8 bytes.
 - If the element is a LOB column descriptor, the element length is 12 bytes.

For information about the column descriptor array or the LOB column descriptor array, see Table 104 on page 731).

Alter Column Attribute

Table 113. Alter Column Attribute Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordheader	0(6)
Column ID		6(2)
Old column definition	Column descriptor ¹	8(8)
New column definition	Column descriptor ¹	16(8)
<i>Total Length: 24 bytes plus record length.</i>		

¹For a description of the column descriptor array, see Table 104 on page 731).

Undo Alter Column Attribute

Table 114. Undo Alter Column Attribute Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(32)
Column ID		32(2)
Old column definition	Column descriptor ¹	34(8)
New column definition	Column descriptor ¹	42(8)
<i>Total Length: 50 bytes plus record length.</i>		

¹For a description of the column descriptor array, see Table 104 on page 731).

Insert Record, Rollback Delete Record, Rollback Update Record

These log records are written when rows are inserted into a table, or when a deletion or update is rolled back. Insert Record and Delete Record log records can also be generated during an update, if the location of the record being updated must be changed to accommodate the modified record data. Insert Record log records are Normal log records. Rollback Delete records and rollback update records are Compensation log records.

Table 115. Insert Record, Rollback Delete Record, Rollback Update Record Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Record Length	unsigned short	6(2)
RID	char[]	8(46)
Free space	unsigned short	14(2)
Record offset	unsigned short	16(2)
Record header and data	variable	18(variable)
<i>Total Length: 18 bytes plus record length</i>		

Record Header and Data Details:

Record header

- 4 bytes
- Record type (unsigned char, 1 byte).
- Reserved (char, 1 byte)
- Record length (unsigned short, 2 bytes)

Record

- Variable length
- Record type (unsigned char, 1 byte).
- Reserved (char, 1 byte)
- The rest of the record is dependent upon the record type and the table descriptor record defined for the table.
- 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 contains user data if the record type in the record header is one of:

- 0x00
- 0x10
- 0x04

and the record type in the record is one of:

- 0x01
- 0x02

Otherwise the record is an internal record.

Formatted User Data record for a table without VALUE COMPRESSION

For records formatted without VALUE COMPRESSION, 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

Data Manager Log Records

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.

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 116. Table Descriptor Record Structure

record type	number of columns	column structure	LOB information
		<ul style="list-style-type: none"> • field type • length • null flag • field offset 	

There is also datalinks information following the LOB information as per earlier comments.

Note: For more information, see Table 103 on page 731.

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 117. 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 115 on page 736.

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

Records formatted with VALUE COMPRESSION consist 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 consume only one byte of disk space which is used for attribute byte. The attribute byte indicates that the column data is compressed, for example, the data value is known but is not stored on disk. The high bit (0x8000) in the offset is used to indicate that the accessed data is an attribute byte. (Only 15 bits are used to represent the offset of the corresponding column data.)
2. For regular column data, the column data follows the offset array. There will not be any attribute byte or any length indicator present.
3. Accessed data can take 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 118. 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 115 on page 736.

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 Record 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 Record 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 to Empty Page log records

Data Manager Log Records

and Delete Record to Empty Page log records are Normal log records. Rollback Delete Record log records and Rollback Insert Record log records are Compensation log records.

Table 119. Rollback Insert Record to Empty Page

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	unsigned short	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)
<i>Total length: 20 bytes</i>		

Table 120. Insert Record to Empty Page, Rollback Delete Record to Empty Page, Delete Record to Empty page

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Record Length	unsigned short	6(2)
RID	char[]	8(46)
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)
<i>Total Length: 22 bytes plus Record length</i>		

Note: For Record Header and Data Details, see Table 115 on page 736.

Update Record

The update record log record is written when a row is updated and its storage location remains the same. There are two available record formats; they are identical to the insert record (also the same as the delete log record) log records (see "Insert Record, Rollback Delete Record, Rollback Update Record" on page 736). 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 121. Update Record Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
New Record Length	unsigned short	6(2)
RID	char[]	8(46)
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)

Table 121. Update Record Log Record Structure (continued)

Description	Type	Offset (Bytes)
Old Record Length	unsigned short	variable(2)
RID	char[]	variable(46)
Free space	unsigned short	variable(2)
Record offset	unsigned short	variable(2)
New record header and data	variable	variable(variable)
<i>Total Length: 36 bytes plus 2 Record lengths</i>		

Rename of a Table or Schema Log Record

The Rename of a Table Schema Log Record is written when a table or schema name is modified.

Table 122. Rename of a Table or Schema Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
<i>Total Length: 6 bytes</i>		

The Rename of a Table or Schema Log Record does not contain information regarding the old and new names of a table or schema object. Separate insert, update, and delete log records associated with operations on the system catalog tables are generated when a table or schema renaming takes place.

Undo Rename of a Table or Schema Log Record

The Undo Rename of a Table Schema Log Record is written when a table or schema name modification is rolled back.

Table 123. Undo Rename of a Table or Schema Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(32)
<i>Total Length: 32 bytes</i>		

The Rename of a Table or Schema Log Record does not contain information regarding the old and new names of a table or schema object. Separate insert, update, and delete log records associated with operations on the system catalog tables are generated when a table or schema renaming takes place.

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 VARCHARIC 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 124. Long Field Manager Log Record Header (LongFieldLogRecordHeader)

Description	Type	Offset (Bytes)
Originator code (component identifier = 3)	unsigned char	0(1)
Operation type (See Table 125.)	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)
<i>Total Length: 10 bytes</i>		

Notes:

1. Table space ID of the data object
2. Object ID of the data object

Table 125. 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 126. 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)

Table 126. Add/Delete/Non-update Long Field Record Log Record Structure (continued)

Description	Type	Offset (Bytes)
Column identifier ²	unsigned short	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 recorded as the first 4 bytes of long field descriptor (LF descriptor), which is logged in the following insert/delete/update log record as part of formatted user data record). 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 (UTC), and mark the time (in seconds) since January 01, 1970.

Normal commit

This log record is written for a transaction in a single-node environment, or in a multiple nodes environment, while the transaction only affects one node. The log record is written when a transaction commits after one of the following events:

1. A user has issued a COMMIT
2. An implicit commit occurs during a CONNECT RESET

Table 127. Normal Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
Authorization identifier length ¹ (if the log record is marked as propagatable)	unsigned short	28(2)
Authorization identifier of the application ¹ (if the log record is marked as propagatable)	char []	30(variable ²)
<i>Total length: 30 bytes plus variable propagatable (28 bytes nonpropagatable)</i>		

Notes:

1. If the log record is marked as propagatable
2. Variable based on Authorization identifier length

Heuristic Commit

This log record is written when an indoubt transaction is committed.

Table 128. Heuristic Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	sqluint64	20(8)
Authorization identifier length ¹ (if the log record is marked as propagatable)	unsigned short	28(2)
Authorization identifier of the application ¹ (if the log record is marked as propagatable)	char []	30(variable ²)
<i>Total length: 30 bytes plus variable propagatable (28 bytes nonpropagatable)</i>		

Notes:

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

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 129. 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 length ¹ (if the log record is marked as propagatable)	unsigned short	variable(2)
Authorization identifier of the application ¹ (if the log record is marked as propagatable)	char []	variable(variable ²)
<i>Total length: variable</i>		

Notes:

1. TNL defines the nodes except for the coordinator node that involved in a transaction
2. Variable based on authorization identifier length

MPP Subordinator Commit

This log record is written on a subordinator node in MPP.

Table 130. 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)

Table 130. MPP Subordinator Commit Log Record Structure (continued)

Description	Type	Offset (Bytes)
Reserved	unsigned short	48(variable)
Authorization identifier length ¹ (if the log record is marked as propagatable)	unsigned short	variable(2)
Authorization identifier of the application ¹ (if the log record is marked as propagatable)	char []	variable(variable ²)
<i>Total length: 48 bytes plus variable</i>		

Notes:

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

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 131. Normal Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier length ¹ (if the log record is marked as propagatable)	unsigned short	20(2)
Authorization identifier of the application ¹ (if the log record is marked as propagatable)	char []	22(variable ²)
<i>Total length: 22 bytes plus variable propagatable (20 bytes nonpropagatable)</i>		

Notes:

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

Heuristic Abort

This log record is written when an indoubt transaction is aborted.

Table 132. Heuristic Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier length ¹ (if the log record is marked as propagatable)	unsigned short	20(2)
Authorization identifier of the application ¹ (if the log record is marked as propagatable)	char []	22(variable ²)
<i>Total length: 22 bytes plus variable propagatable (20 bytes nonpropagatable)</i>		

Transaction Manager Log Records

Notes:

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

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 133. 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)
<i>Total Length: 30 bytes plus variables propagatable (28 bytes plus pending list entries non-propagatable)</i>		

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 134. 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)
<i>Total Length: 22 bytes plus variables propagatable (20 bytes plus pending list entries non-propagatable)</i>		

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 135. 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)
Reserve	sqluint32	variable(2)
XA identifier of the transaction	SQLXA_XID ¹	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)
<i>Total length: 40 bytes plus variables</i>		

Note: 1. For details on the SQLXA_XID log record type, see SQLXA_XID data structure. in the DB2 Administrative APIs reference.

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 136. 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)
MPP identifier of the transaction	SQLP_GXID ¹	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)
<i>Total Length: 152 bytes plus variable</i>		

Transaction Manager Log Records

Note: 1.The SQLP-GXID log record is used to identify transactions in MPP environment.

Table 137. Fields in the SQLP-GXID Structure

Field Name	Data Type	Description
FORMATID	INTEGER	GXID format ID
GXID_LENGTH	INTEGER	Length of GXID
BQAL_LENGTH	INTEGER	Length of the branch identifier
DATA	CHAR(8)	First 2 bytes contain the node number; remainder is the transaction ID

TM Prepare

This log record is written for DB2 coordinated transactions in a single-partition environment or on the coordinator node in MPP, where the database is acting as the TM database. The log record is written to mark the preparation of the transaction as part of a two-phase commit.

Table 138. TM 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)
Reserve	sqluint32	variable(2)
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 []	variable(20)
Authorization identifier	char []	variable(variable)
Synclog information	variable	variable(variable)
<i>Total length: 40 bytes plus variables</i>		

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 a 6-byte log sequence number (*compls*, stored in the log record header starting at offset 20). Under certain scenarios, the backout free log record also contains log data, starting at offset 26, which is same as the data logged in corresponding data manager log records. When this log record is read during rollback (following an aborted transaction), *compls* marks the next log record to be compensated.

Table 139. Backout free Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Complsn	SQLP_LSN	20(6)
Log data ¹	variable	variable
<i>Total Length: 26 bytes plus variables</i>		

Note: 1. Only applied in certain scenarios, and when used, the length of the entire log record in the log header is more than 26 bytes.

Utility Manager Log Records

The utility manager produces log records associated with the following DB2 utilities:

- Migration
- Load
- Backup
- Table space rollforward.

The log records signify the beginning or the end of the requested activity. Only propogatable log records for these utilities are documented.

Migration Begin

This log record is associated with the beginning of catalog migration.

Table 140. 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)
<i>Total Length: 34 bytes</i>		

Migration End

This log record is associated with the successful completion of catalog migration.

Table 141. 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)
<i>Total Length: 32 bytes</i>		

Load Start

This log record is associated with the beginning of a load.

It is the only Load log record that is propogatable. It is written at the beginning of the Load phase. This log record should not be confused with other types of Load Start records written at the beginning of a Setup phase which is not propogatable.

For the purpose of log record propogation, it is recommended that after reading a Log Start log record you not continue to propogate log records for the specific

Utility Manager Log Records

table to a target table. After a Load Start log record, all propagatable log records that belong to the table being loaded can be ignored regardless of the transaction boundary, until such a time that a cold restart has taken place. A cold restart is required to synchronize the source and target tables.

Table 142. 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	sqluint32	28(4)
Object pool list	variable	32(variable)
<i>Total length: 32 bytes plus variable</i>		

Backup End

This log record is associated with the end of a successful backup.

Table 143. Backup End Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Backup end time	sqluint64	20(8)
<i>Total Length: 28 bytes</i>		

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 144. Table Space Rolled Forward Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Table space identifier	sqluint32	20(4)
<i>Total length: 24 bytes</i>		

Table Space Roll Forward to Point in Time Starts

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 145. Table Space Roll Forward to Point in Time Starts Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time stamp for this log record	sqluint64	20(8)
Time stamp to which table spaces are being rolled forward	sqluint32	28(4)
Number of pools being rolled forward	sqluint32	32(4)
<i>Total length: 36 bytes</i>		

Table Space Roll Forward to Point in Time 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 146. Table Space Roll Forward to Point in Time Ends Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time stamp for this log record	sqluint64	20(8)
Time stamp to which table spaces were rolled forward	sqluint32	28(4)
A flag whose value is TRUE if the roll forward was successful, or FALSE if the roll forward was canceled.	sqluint32	32(4)
<i>Total length: 36 bytes</i>		

Two timestamp fields are required to provide adequate precision so that event log event timing can be differentiated. The first timestamp uses 8 bytes to indicate the time when the log was written to a precision of seconds. The first 4 bytes of this timestamp indicate the seconds portion. Since many actions can take place in one second, to understand the ordering of events it is necessary to have further precision. The second timestamp field provides 4 bytes that are used to represent nanoseconds. If the log record timestamps of two log records are identical, the additional 4 byte timestamp field can be used to determine the ordering of the associated log events.

Related reference:

- “db2ReadLog - Extracts log records” on page 254

Application migration

Administrative APIs and application migration

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.

Administrative APIs and application migration

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**.

Related reference:

- “Changed APIs and data structures” on page 752

Changed APIs and data structures

Table 147. 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)
sqledosd (V8.1)	Open Database Directory Scan	db2DbDirOpenScan (V8.2)
sqledgne (V8.1)	Get Next Database Directory Entry	db2DbDirGetNextEntry (V8.2)
sqledcls (V8.1)	Close Database Directory Scan	db2DbDirCloseScan (V8.2)
sqlestart (V5)	Start Database Manager	db2InstanceStart (V8)
sqlestop (V5)	Stop Database Manager	db2InstanceStop (V8)
sqlepar (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)
sqlfdfs (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)

Changed DB2 APIs and data structures

Table 147. Back-level supported APIs and data structures (continued)

API or Data Structure (Version)	Descriptive Name	New API or Data Structure (Version)
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)
sqluexpr	Export	db2Export (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)
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)
sqluimpr	Import	db2Import (V8)
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)
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)
SQLEDINFO (v8.1)	Get Next Database Directory Entry data structure	db2DbDirInfo (V8.2)
SQLUEXPT-OUT	Export output structure	db2ExportOut (V8.2)
SQLUHINFO and SQLUHADM (V5)	History file data structures	db2HistData (V6)
SQLUIMPT-IN	Import input structure	db2ImportIn (V8.2)
SQLUIMPT-OUT	Import output structure	db2ImportOut (V8.2)
SQLULOAD-IN (V7)	Load input structure	db2LoadIn (V8)
SQLULOAD-OUT (V7)	Load output structure	db2LoadOut (V8)
db2DbDirInfo (V8.2)	Get Next Database Directory Entry data structure	db2DbDirInfoV9 (V9.1)
db2DbDirNextEntryStruct (V8.2)	Get Next Database Directory Entry data structure	db2DbDirNextEntryStructV9 (V9.1)

Changed DB2 APIs and data structures

Table 147. Back-level supported APIs and data structures (continued)

API or Data Structure (Version)	Descriptive Name	New API or Data Structure (Version)
db2gDbDirNextEntryStruct (V8.2)	Get Next Database Directory Entry data structure	db2gDbDirNextEntryStrV9 (V9.1)

Table 148. Back-level unsupported APIs and data structures

Name	Descriptive Name	API or data structure supported in V9
sqlufrol/sqlgfrol	Roll Forward Database (DB2 Version 1.1)	db2Rollforward
sqluprfrw	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
sqlxphqr	List an Indoubt Transaction	db2XaListIndTrans
SQLXA-RECOVER	Transaction API structure	db2XaRecoverStruct

Related reference:

- “Administrative APIs and application migration” on page 751

REXX APIs

Calling DB2 APIs in REXX

Use the SQLDBS routine to call DB2 APIs with the following syntax:

```
CALL SQLDBS 'command string'
```

If a DB2 API you want to use cannot be called using the SQLDBS routine, you may still call the API by calling the DB2 command line processor (CLP) from within the REXX application. However, because the DB2 CLP directs output either to the standard output device or to a specified file, your REXX application cannot directly access the output from the called DB2 API, nor can it easily make a determination as to whether the called API is successful or not. The SQLDB2 API provides an interface to the DB2 CLP that provides direct feedback to your REXX application on the success or failure of each called API by setting the compound REXX variable, SQLCA, after each call.

You can use the SQLDB2 routine to call DB2 APIs using the following syntax:

```
CALL SQLDB2 'command string'
```

where 'command string' is a string that can be processed by the command-line processor (CLP).

Calling a DB2 API using SQLDB2 is equivalent to calling the CLP directly, except for the following:

- The call to the CLP executable is replaced by the call to SQLDB2 (all other CLP options and parameters are specified the same way).
- The REXX compound variable SQLCA is set after calling the SQLDB2 but is not set after calling the CLP executable.

Changed DB2 APIs and data structures

- The default display output of the CLP is set to off when you call SQLDB2, whereas the display is set to on output when you call the CLP executable. Note that you can turn the display output of the CLP to on by passing the +o or the -o- option to the SQLDB2.

Because the only REXX variable that is set after you call SQLDB2 is the SQLCA, you only use this routine to call DB2 APIs that do not return any data other than the SQLCA and that are not currently implemented through the SQLDBS interface. Thus, only the following DB2 APIs are supported by SQLDB2:

- Activate Database
- Add Node
- Bind for DB2 Version 1⁽¹⁾ ⁽²⁾
- Bind for DB2 Version 2 or 5⁽¹⁾
- Create Database at Node
- Drop Database at Node
- Drop Node Verify
- Deactivate Database
- Deregister
- Load⁽³⁾
- Load Query
- Precompile Program⁽¹⁾
- Rebind Package⁽¹⁾
- Redistribute Database Partition Group
- Register
- Start Database Manager
- Stop Database Manager

Notes on DB2 APIs Supported by SQLDB2:

1. These commands require a CONNECT statement through the SQLDB2 interface. Connections using the SQLDB2 interface are not accessible to the SQLEXEC interface and connections using the SQLEXEC interface are not accessible to the SQLDB2 interface.
2. Is supported on Windows-based platforms through the SQLDB2 interface.
3. The optional output parameter, pLoadInfoOut for the Load API is not returned to the application in REXX.

Note: Although the SQLDB2 routine is intended to be used only for the DB2 APIs listed above, it can also be used for other DB2 APIs that are not supported through the SQLDBS routine. Alternatively, the DB2 APIs can be accessed through the CLP from within the REXX application.

Related concepts:

- “Embedded SQL statements in REXX applications” in *Developing Embedded SQL Applications*

Change Isolation Level

Changes the way that DB2 isolates data from other processes while a database is being accessed. This API can only be called from a REXX application.

Authorization:

None

Required connection:

Change Isolation Level

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 concepts:

- “Calling DB2 APIs in REXX” on page 754

Related reference:

- “REXX samples” in *Samples Topics*
- “CHANGE ISOLATION LEVEL command” in *Command Reference*

DB2 Database technical information

Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
 - Topics
 - Help for DB2 tools
 - Sample programs
 - Tutorials
- DB2 books
 - PDF files (downloadable)
 - PDF files (from the DB2 PDF CD)
 - printed books
- Command line help
 - Command help
 - Message help
- Sample programs

IBM® periodically makes documentation updates available. If you access the online version on the DB2 Information Center at ibm.com®, you do not need to install documentation updates because this version is kept up-to-date by IBM. If you have installed the DB2 Information Center, it is recommended that you install the documentation updates. Documentation updates allow you to update the information that you installed from the *DB2 Information Center CD* or downloaded from Passport Advantage as new information becomes available.

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and Redbooks™ online at [ibm.com](http://www.ibm.com). Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

Documentation feedback: We value your feedback on the DB2 documentation. If you have suggestions for how we can improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

Related concepts:

- “Features of the DB2 Information Center” in *Online DB2 Information Center*
- “Sample files” in *Samples Topics*

Related tasks:

- “Invoking command help from the command line processor” in *Command Reference*
- “Invoking message help from the command line processor” in *Command Reference*
- “Updating the DB2 Information Center installed on your computer or intranet server” on page 762

Related reference:

- “DB2 technical library in hardcopy or PDF format” on page 757

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. DB2 Version 9 manuals in PDF format can be downloaded from www.ibm.com/software/data/db2/udb/support/manualsv9.html.

Although the tables identify books available in print, the books might not be available in your country or region.

The information in these books is fundamental to all DB2 users; you will find this information useful whether you are a programmer, a database administrator, or someone who works with DB2 Connect or other DB2 products.

Table 149. DB2 technical information

Name	Form Number	Available in print
<i>Administration Guide: Implementation</i>	SC10-4221	Yes

Change Isolation Level

Table 149. DB2 technical information (continued)

Name	Form Number	Available in print
<i>Administration Guide: Planning</i>	SC10-4223	Yes
<i>Administrative API Reference</i>	SC10-4231	Yes
<i>Administrative SQL Routines and Views</i>	SC10-4293	No
<i>Call Level Interface Guide and Reference, Volume 1</i>	SC10-4224	Yes
<i>Call Level Interface Guide and Reference, Volume 2</i>	SC10-4225	Yes
<i>Command Reference</i>	SC10-4226	No
<i>Data Movement Utilities Guide and Reference</i>	SC10-4227	Yes
<i>Data Recovery and High Availability Guide and Reference</i>	SC10-4228	Yes
<i>Developing ADO.NET and OLE DB Applications</i>	SC10-4230	Yes
<i>Developing Embedded SQL Applications</i>	SC10-4232	Yes
<i>Developing SQL and External Routines</i>	SC10-4373	No
<i>Developing Java Applications</i>	SC10-4233	Yes
<i>Developing Perl and PHP Applications</i>	SC10-4234	No
<i>Getting Started with Database Application Development</i>	SC10-4252	Yes
<i>Getting started with DB2 installation and administration on Linux and Windows</i>	GC10-4247	Yes
<i>Message Reference Volume 1</i>	SC10-4238	No
<i>Message Reference Volume 2</i>	SC10-4239	No
<i>Migration Guide</i>	GC10-4237	Yes
<i>Net Search Extender Administration and User's Guide</i>	SH12-6842	Yes
Note: HTML for this document is not installed from the HTML documentation CD.		
<i>Performance Guide</i>	SC10-4222	Yes
<i>Query Patroller Administration and User's Guide</i>	GC10-4241	Yes
<i>Quick Beginnings for DB2 Clients</i>	GC10-4242	No
<i>Quick Beginnings for DB2 Servers</i>	GC10-4246	Yes
<i>Spatial Extender and Geodetic Data Management Feature User's Guide and Reference</i>	SC18-9749	Yes
<i>SQL Guide</i>	SC10-4248	Yes

Table 149. DB2 technical information (continued)

Name	Form Number	Available in print
<i>SQL Reference, Volume 1</i>	SC10-4249	Yes
<i>SQL Reference, Volume 2</i>	SC10-4250	Yes
<i>System Monitor Guide and Reference</i>	SC10-4251	Yes
<i>Troubleshooting Guide</i>	GC10-4240	No
<i>Visual Explain Tutorial</i>	SC10-4319	No
<i>What's New</i>	SC10-4253	Yes
<i>XML Extender Administration and Programming</i>	SC18-9750	Yes
<i>XML Guide</i>	SC10-4254	Yes
<i>XQuery Reference</i>	SC18-9796	Yes

Table 150. DB2 Connect-specific technical information

Name	Form Number	Available in print
<i>DB2 Connect User's Guide</i>	SC10-4229	Yes
<i>Quick Beginnings for DB2 Connect Personal Edition</i>	GC10-4244	Yes
<i>Quick Beginnings for DB2 Connect Servers</i>	GC10-4243	Yes

Table 151. WebSphere® Information Integration technical information

Name	Form Number	Available in print
<i>WebSphere Information Integration: Administration Guide for Federated Systems</i>	SC19-1020	Yes
<i>WebSphere Information Integration: ASNCLP Program Reference for Replication and Event Publishing</i>	SC19-1018	Yes
<i>WebSphere Information Integration: Configuration Guide for Federated Data Sources</i>	SC19-1034	No
<i>WebSphere Information Integration: SQL Replication Guide and Reference</i>	SC19-1030	Yes

Note: The DB2 Release Notes provide additional information specific to your product's release and fix pack level. For more information, see the related links.

Related concepts:

- "Overview of the DB2 technical information" on page 756
- "About the Release Notes" in *Release notes*

Related tasks:

- "Ordering printed DB2 books" on page 760

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation CD* are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation CD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation CD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at <http://publib.boulder.ibm.com/infocenter/db2help/>.

Procedure:

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at <http://www.ibm.com/shop/publications/order>. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
 - Locate the contact information for your local representative from one of the following Web sites:
 - The IBM directory of world wide contacts at www.ibm.com/planetwide
 - The IBM Publications Web site at <http://www.ibm.com/shop/publications/order>. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
 - When you call, specify that you want to order a DB2 publication.
 - Provide your representative with the titles and form numbers of the books that you want to order.

Related concepts:

- "Overview of the DB2 technical information" on page 756

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 757

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure:

To invoke SQL state help, open the command line processor and enter:

? *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, and ? 08 displays help for the 08 class code.

Related tasks:

- “Invoking command help from the command line processor” in *Command Reference*
- “Invoking message help from the command line processor” in *Command Reference*

Accessing different versions of the DB2 Information Center

For DB2 Version 9 topics, the DB2 Information Center URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>.

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

Related tasks:

- “Setting up access to DB2 contextual help and documentation” in *Administration Guide: Implementation*

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

Procedure:

To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
 - To add a new language to the list, click the **Add...** button.

Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.

- To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

To display topics in your preferred language in a Firefox or Mozilla browser:

1. Select the **Tools** —> **Options** —> **Languages** button. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.

Change Isolation Level

- To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
 - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Related concepts:

- “Overview of the DB2 technical information” on page 756

Updating the DB2 Information Center installed on your computer or intranet server

If you have a locally-installed DB2 Information Center, updated topics can be available for download. The 'Last updated' value found at the bottom of most topics indicates the current level for that topic.

To determine if there is an update available for the entire DB2 Information Center, look for the 'Last updated' value on the Information Center home page. Compare the value in your locally installed home page to the date of the most recent downloadable update at <http://www.ibm.com/software/data/db2/udb/support/icupdate.html>. You can then update your locally-installed Information Center if a more recent downloadable update is available.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to determine if update packages are available from IBM.

Note: Updates are also available on CD. For details on how to configure your Information Center to install updates from CD, see the related links.

If update packages are available, use the Update feature to download the packages. (The Update feature is only available in stand-alone mode.)

3. Stop the stand-alone Information Center, and restart the DB2 Information Center service on your computer.

Procedure:

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center service.
 - On Windows®, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
 - On Linux, enter the following command:
`/etc/init.d/db2icdv9 stop`
2. Start the Information Center in stand-alone mode.

- On Windows:
 - a. Open a command window.
 - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the C:\Program Files\IBM\DB2 Information Center\Version 9 directory.
 - c. Run the help_start.bat file using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>\doc\bin\help_start.bat
```
- On Linux:
 - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the /opt/ibm/db2ic/V9 directory.
 - b. Run the help_start script using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>/doc/bin/help_start
```

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the Update button (🔄). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click **Install Updates**.
5. After the download and installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center.
 - On Windows, run the help_end.bat file using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>\doc\bin\help_end.bat
```

Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use Ctrl-C or any other method to terminate help_start.bat.
 - On Linux, run the help_end script using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>/doc/bin/help_end
```

Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the help_start script.
7. Restart the DB2 Information Center service.
 - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
 - On Linux, enter the following command:

```
/etc/init.d/db2icdv9 start
```

The updated DB2 Information Center displays the new and updated topics.

Related concepts:

- “DB2 Information Center installation options” in *Quick Beginnings for DB2 Servers*

Related tasks:

Change Isolation Level

- “Installing the DB2 Information Center using the DB2 Setup wizard (Linux)” in *Quick Beginnings for DB2 Servers*
- “Installing the DB2 Information Center using the DB2 Setup wizard (Windows)” in *Quick Beginnings for DB2 Servers*

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin:

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2help/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials:

To view the tutorial, click on the title.

Native XML data store

Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

Visual Explain Tutorial

Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

Related concepts:

- “Visual Explain overview” in *Administration Guide: Implementation*

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation

Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site

Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at <http://www.ibm.com/software/data/db2/udb/support.html>

Related concepts:

- “Introduction to problem determination” in *Troubleshooting Guide*
- “Overview of the DB2 technical information” on page 756

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

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

Change Isolation Level

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 improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may

vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

Company, product, or service names identified in the documents of the DB2 Version 9 documentation library may be trademarks or service marks of International Business Machines Corporation or other companies. Information on the trademarks of IBM Corporation in the United States, other countries, or both is located at <http://www.ibm.com/legal/copytrade.shtml>.

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 documentation library:

Microsoft, Windows, Windows NT[®], and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel[®], Itanium[®], Pentium[®], and Xeon[®] are trademarks of Intel Corporation in the United States, other countries, or both.

Change Isolation Level

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

- abnormal termination
 - restart API 64
- Activate Database API 383
- Add a snapshot request API 28
- Add Contact API 24
- Add Contact Group API 26
- add long field record log record 725
- Add Node API 388
- Administration Message Write API 30
- alter table add columns log record 725
- alter table attribute log record 725
- anyorder file type modifier 205
- APIs
 - back level 752
 - Change Isolation Level (REXX) 755
 - Compress 710
 - db2AddContact 24
 - db2AddContactGroup 26
 - db2AddSnapshotRequest 28
 - db2AdminMsgWrite 30
 - db2ArchiveLog 32
 - db2AutoConfig 35
 - db2AutoConfigFreeMemory 40
 - db2Backup 41
 - db2CfgGet 49
 - db2CfgSet 53
 - db2ConvMonStream 57
 - db2DatabasePing 60
 - db2DatabaseQuiesce 62
 - db2DatabaseRestart 64
 - db2DatabaseUnquiesce 67
 - db2DropContact 77
 - db2DropContactGroup 79
 - db2Export 81
 - db2GetAlertCfg 95
 - db2GetAlertCfgFree 100
 - db2GetContactGroup 101
 - db2GetContactGroups 103
 - db2GetContacts 105
 - db2GetHealthNotificationList 107
 - db2GetRecommendations 109
 - db2GetRecommendationsFree 112
 - db2GetSnapshot 113
 - db2GetSnapshotSize 117
 - db2GetSyncSession 120
 - db2HADRStart 121
 - db2HADRStop 123
 - db2HADRTakeover 125
 - db2HistoryCloseScan 127
 - db2HistoryGetEntry 129
 - db2HistoryOpenScan 132
 - db2HistoryUpdate 136
 - db2Import 139
 - db2Inspect 164
 - db2InstanceQuiesce 170
 - db2InstanceStart 173
 - db2InstanceStop 179
 - db2InstanceUnquiesce 183
 - db2LdapCatalogDatabase 185
 - db2LdapCatalogNode 188
- APIs (*continued*)
 - db2LdapDeregister 190
 - db2LdapRegister 192
 - db2LdapUncatalogDatabase 196
 - db2LdapUncatalogNode 198
 - db2LdapUpdate 200
 - db2LdapUpdateAlternateServerForDB 203
 - db2Load 205
 - db2LoadQuery 239
 - db2MonitorSwitches 246
 - db2Prune 249
 - db2QuerySatelliteProgress 252
 - db2ReadLog 254
 - db2ReadLogNoConn 259
 - db2ReadLogNoConnInit 262
 - db2ReadLogNoConnTerm 264
 - db2Recover 265
 - db2Reorg 271
 - db2ResetAlertCfg 279
 - db2ResetMonitor 281
 - db2Restore 284
 - db2Rollforward 296
 - db2Runstats 306
 - db2SelectDB2Copy 316
 - db2SetSyncSession 318
 - db2SetWriteForDB 320
 - db2SpmListIndTrans 322
 - db2SyncSatellite 325
 - db2SyncSatelliteStop 326
 - db2SyncSatelliteTest 327
 - db2UpdateAlertCfg 328
 - db2UpdateAlternateServerForDB 334
 - db2UpdateContact 336
 - db2UpdateContactGroup 338
 - db2UpdateHealthNotificationList 340
 - db2UtilityControl 342
 - db2VendorGetNextObj 694
 - db2VendorQueryApiVersion 693
 - db2XaGetInfo 344
 - db2XaListIndTrans 345
 - Decompress 715
 - GetMaxCompressedSize 712
 - GetSavedBlock 709
 - InitCompression 708
 - InitDecompression 714
 - plug-in 627, 639
 - precompiler customization 723
 - security plug-in 625, 630, 632, 633, 637, 638, 639, 646, 648, 649, 651, 653, 655, 658, 659, 660, 661, 664, 665, 667
 - sqlabndx 350
 - sqlaintp 353
 - sqlaprep 355
 - sqlarbind 358
 - sqlbctcq 361
 - sqlbctsq 362
 - sqlbftcq 363
 - sqlbftpq 365
 - sqlbgtss 367
 - sqlbmtsq 369
 - sqlbotcq 371
 - sqlbotsq 373

APIs (continued)

- sqlbstpq 375
- sqlbstsc 377
- sqlbtcp 379
- sqlcspqy 381
- sqle_activate_db 383
- sqle_deactivate_db 386
- sqleaddn 388
- sqleatcp 391
- sqleatin 394
- sqleAttachToCtx 397
- sqleBeginCtx 398
- sqlecadb 400
- sqlecran 406
- sqlecrea 408
- sqlectnd 415
- sqledcgd 419
- sqledcls 69
- sqleDetachFromCtx 422
- sqledgne 71
- sqledosd 75
- sqledpan 423
- sqledrpd 425
- sqledrpn 427
- sqledtin 429
- sqleEndCtx 430
- sqlefmem 432
- sqlefrce 433
- sqlegdad 436
- sqlegdcl 438
- sqlegdel 439
- sqlegdge 441
- sqlegdgt 443
- sqlegdsc 445
- sqleGetCurrentCtx 447
- sqlegins 448
- sqleInterruptCtx 450
- sqleintr 451
- sqleisig 453
- sqlemgdb 455
- sqlencls 457
- sqlengne 458
- sqlenops 460
- sqleqryc 462
- sqleqryi 464
- sqlsact 466
- sqlsdeg 468
- sqlsetc 470
- sqlseti 473
- sqleSetTypeCtx 476
- sqleuncd 478
- sqleuncn 480
- sqlgaddr 482
- sqlgdref 483
- sqlgmcpy 484
- sqlogstt 485
- sqluadcu 487
- sqludrdr 489
- sqluexpr 81
- sqlugrpn 493
- sqlugtpi 496
- sqluimpr 139
- sqluvdel 691
- sqluvend 689
- sqluvget 685
- sqluvint 680
- sqluvput 687

APIs (continued)

- sqluvqdp 498
- sqlxhfrg 501
- sqlxphcm 502
- sqlxphrl 504
- summary 1
- TermCompression 713
- TermDecompression 716
- application design
 - installing signal handler routine 453
 - pointer manipulation 482
 - providing pointer manipulation 483, 484
 - setting collating sequence 408
- application migration 751
- application programming interfaces (API)
 - syntax for REXX 754
- applications
 - access through database manager 350
- Archive Active Log API 32
- Asynchronous Read Log API 254
- Attach and Change Password API 391
- Attach API 394
- Attach to Context API 397
- authentication
 - GSS-API 592
 - ID/password 592
 - Kerberos 592
 - plug-ins
 - API for checking if authentication ID exists 667
 - API for cleaning client authentication resources 648
 - API for cleaning up resources 660
 - API for getting authentication IDs 665
 - API for initializing a client authentication plug-in 646
 - API for initializing server authentication 661
 - API for validating passwords 655
 - clean up server authentication 664
 - deploying 606, 607, 611
 - for initializing a client authentication plug-in 646
 - library locations 597
 - user ID/ password 639
 - security plug-in 592
 - two-part user IDs 600
- Autoconfigure API 35

B

- backout free log record 725
- backup and restore vendor products 591, 671
- Backup database API 41
- backup end log record 725
- binarynumerics file type modifier 205
- Bind API
 - sqlabndx 350
- binding
 - application programs to databases 350
 - defaults 350
 - errors 408

C

- catalog database API 400
- catalog database LDAP entry API 185
- catalog DCS database API 436
- catalog node API 415
- catalog node LDAP entry API 188
- change database comment API 419

- change isolation level REXX API 755
- chardel file type modifier
 - export 81
 - import 139
 - load 205
- close database directory scan API 69
- close DCS directory scan API 438
- close history file scan API 127
- close node directory scan API 457
- close table space container query API 361
- close table space query API 362
- COBOL language
 - pointer manipulation 482, 483, 484
- code page file type modifier 205
- code pages
 - Export API 81
 - Import API 139
- codel file type modifier
 - export 81
 - import 139
 - load 205
- collating sequences
 - user-defined 408
- columns
 - specifying for import 139
- command line processor (CLP)
 - calling from REXX application 754
- comments
 - database, changing 419
- commit an indoubt transaction API 502
- compound file type modifier 139
- COMPR_CB structure 721
- COMPR_DB2INFO structure 717
- COMPR_PIINFO structure 719
- Compress a block of data API 710
- compression
 - Stop the compression library API 713
 - Stop the decompression library API 716
- Compression plug-in interface 591, 705
- concurrency control 755
- contacting IBM 771
- convert monitor stream API 57
- copy memory API 484
- create and attach to an application context API 398
- create database API
 - description 408
- create database at node API 406
- create index log record 725
- create table log record 725
- customizing
 - database management 591

D

- Data Links Manager
 - log records 725
- DATA structure 704
- data structures
 - COMPR_CB 721
 - COMPR_DB2INFO 717
 - COMPR_PIINFO 719
 - DB2-INFO 696
 - db2HistData 507
 - INIT-OUTPUT 703
 - RETURN-CODE 705
 - sql_authorizations 513
 - SQL-DIR-ENTRY 515

- data structures (*continued*)
 - SQLB-TBS-STATS 517
 - SQLB-TBSCONTQRY-DATA 519
 - SQLB-TBSPQRY-DATA 521
 - SQLCA 526
 - SQLCHAR 527
 - SQLDA 528
 - SQLDCOL 530
 - SQLE-ADDN-OPTIONS 533
 - SQLE-CLIENT-INFO 535
 - SQLE-CONN-SETTING 538
 - SQLE-NODE-CPIC 541
 - SQLE-NODE-IPXSPX 542
 - SQLE-NODE-LOCAL 543
 - SQLE-NODE-NETB 544
 - SQLE-NODE-NPIPE 545
 - SQLE-NODE-STRUCT 546
 - SQLE-NODE-TCPIP 548
 - SQLEDBTERRITORYINFO 563
 - SQLENINFO 564
 - SQLETSDESC 549
 - SQLFUPD 567
 - sqllob 574
 - SQLMA 575
 - SQLOPT 579
 - SQLU-LSN 581
 - SQLU-MEDIA-LIST 582
 - SQLU-RLOG-INFO 587
 - SQLUPI 588
 - SQLXA-XID 590
 - used by vendor APIs 671
 - VENDOR-INFO 699
- database configuration file
 - valid entries 567
- Database Connection Services (DCS) directory
 - cataloging entries 436
 - copy entries from 443
 - removing entries 439
 - retrieving entries from 441
- database directories
 - retrieving next entry 71
- database manager
 - log records 725
- Database Quiesce API 62
- Database Unquiesce API 67
- databases
 - binding application programs 350
 - concurrent request processing 755
 - creating 408
 - deleting 425
 - deleting, ensuring recovery with log files 425
 - dropping 425
 - exporting table to a file 81
 - importing file to table 139
 - isolating data 755
- dateformat file type modifier 139, 205
- datesiso file type modifier 81, 139, 205
- DB2 Connect
 - supported connections 436
- DB2 Data Links Manager
 - log records
 - delete group 725
 - delete pgroup 725
 - description 725
 - DLFM prepare 725
 - link file 725

- DB2 Information Center
 - updating 762
 - versions 761
 - viewing in different languages 761
- DB2-INFO structure 696
- db2AddContact API 24
- db2AddContactGroup API 26
- db2AdminMsgWrite API 30
- db2ArchiveLog API 32
- db2AutoConfig API 35
- db2AutoConfigFreeMemory API 40
- db2Backup API 41
- db2CfgGet API 49
- db2CfgSet API 53
- db2ConvMonStream API 57
- db2DatabasePing API 60
- db2DatabaseQuiesce API 62
- db2DatabaseRestart API 64
- db2DatabaseUnquiesce API 67
- db2DropContact API 77
- db2DropContactGroup API 79
- db2GetAlertCfg API 95
- db2GetAlertCfgFree API 100
- db2GetContactGroup API 101
- db2GetContactGroups API 103
- db2GetContacts API 105
- db2GetHealthNotificationList API 107
- db2GetRecommendations API 109
- db2GetRecommendationsFree API 112
- db2GetSnapshot API 113
- db2GetSnapshotSize API 117
- db2GetSyncSession API 120
- db2HADRStart API 121
- db2HADRStop API 123
- db2HADRTakeover API 125
- db2HistData structure 507
- db2HistoryCloseScan API 127
- db2HistoryGetEntry API 129
- db2HistoryOpenScan API 132
- db2HistoryUpdate API 136
- db2Inspect API 164
- db2InstanceQuiesce API 170
- db2InstanceStart API 173
- db2InstanceStop API 179
- db2InstanceUnquiesce API 183
- db2LdapCatalogDatabase API 185
- db2LdapCatalogNode API 188
- db2LdapDeregister API 190
- db2LdapRegister API 192
- db2LdapUncatalogDatabase API 196
- db2LdapUncatalogNode API 198
- db2LdapUpdate API 200
- db2LdapUpdateAlternateServerForDB 203
- db2Load API 205
- db2LoadQuery API 239
- db2MonitorSwitches API 246
- db2Prune API 249
- db2QuerySatelliteProgress API 252
- db2ReadLog API 254
- db2ReadLogNoConn API 259
- db2ReadLogNoConnInit API 262
- db2ReadLogNoConnTerm API 264
- db2Recover API 265
- db2Reorg API 271
- db2ResetAlertCfg API 279
- db2ResetMonitor API 281
- db2Restore API 284
- db2Rollforward API 296
- db2Runstats API 306
- db2SetSyncSession API 318
- db2SetWriteForDB API 320
- db2SyncSatellite API 325
- db2SyncSatelliteStop API 326
- db2SyncSatelliteTest API 327
- db2UpdateAlertCfg API 328
- db2UpdateAlternateServerForDB API 334
- db2UpdateContact API 336
- db2UpdateContactGroup API 338
- db2UpdateHealthNotificationList API 340
- db2UtilityControl API 342
- db2VendorGetNextObj API 694
- db2VendorQueryApiVersion API 693
- db2XaGetInfo API 344
- db2XaListIndTrans API 345
- Deactivate Database API 386
- debugging
 - security plug-ins 604
- Decompress a block of data API 715
- decplusblank file type modifier 81, 139, 205
- decpct file type modifier 81, 139, 205
- Delete Committed Session API 691
- delete group log record 725
- delete long field record log record 725
- delete pgroup log record 725
- delete record log record 725
- delimiter restrictions
 - moving data 237
- delprioritychar file type modifier 139, 205
- Dereference Address API 483
- Detach and Destroy Application Context API 430
- Detach API 429
- Detach From Context API 422
- directories
 - cataloging 415
 - Database Connection Services
 - retrieving entries from 441
 - Database Connection Services (DCS), cataloging
 - entries 436
 - Database Connection Services (DCS), uncataloging
 - entries 439
 - Database Connection Services, copy entries from 443
 - deleting entries 480
 - local database 75
 - Open DCS Directory Scan API 445
 - retrieving entries from 458
 - retrieving next entry from 71
 - system database 75
 - system database, cataloging 400
 - uncataloging 478
- discontinued APIs and data structures 751
- dldel file type modifier 81, 139, 205
- DLFM (Data Links File Manager)
 - prepare log record 725
- documentation 756, 757
 - terms and conditions of use 765
- Drop Contact API 77
- Drop Contact Group API 79
- Drop Database API 425
- Drop Database at Node API 423
- drop index log record 725
- Drop Node Verify API 427
- DROP statement
 - tables
 - log record 725

dumpfile file type modifier 205

E

error messages

- database description block structure 408
- dropping remote database 425
- during binding 350
- during rollforward 296
- retrieving from SQLCODE field 353
- return codes 353, 485
- security plug-ins 625

Estimate Size Required for db2GetSnapshot Output Buffer API 117

Export API 81

export utility

- file type modifiers 89

exporting

- database tables files 81

exporting data

- file type modifiers for 81
- specifying column names 81

extended database description block 556

F

fastparse file type modifier 205

Fetch Table Space Container Query API 363

Fetch Table Space Query API 365

file type modifiers

- Export API 81
- export utility 89
- Import API 139
- import utility 153
- Load API 205
- load utility 225

Force Application API 433

forcein file type modifier 139, 205

Forget Transaction Status API 501

formatted user data record log record 725

FORTTRAN language

- pointer manipulation 482, 483, 484

Free Autoconfigure Memory API 40

Free db2GetRecommendations Memory API 112

Free Get Alert Configuration API 100

Free Memory API 432

functions

client plug-in

- check if authentication ID exists 667
- clean up client authentication 648
- clean up resources 660
- clean up server authentication 664
- free memory held by token 659
- generate initial credentials 653
- get authentication IDs 665
- get default login context 651
- initialize client authentication 646
- initialize server authentication 661
- process service principal name 658
- remap user ID and password 649
- validate password 655

group plug-in

- check if group exists 637
- clean up 632
- free error message memory 639
- free group list memory 638

functions (*continued*)

group plug-in (*continued*)

- get list of groups 633
- initialization 630

G

generatedignore file type modifier 139, 205

generatedmissing file type modifier 139, 205

generatedoverride file type modifier 205

Get Address API 482

Get Alert Configuration API 95

Get Configuration Parameters API 49

Get Contact Group API 101

Get Contact Groups API 103

Get Contacts API 105

Get Current Context API 447

Get current user's authorities API 487

Get DCS Directory Entries API 443

Get DCS Directory Entry for Database API 441

Get Error Message API 353

Get Health Notification List API 107

Get Information for Resource Manager API 344

Get Instance API 448

Get Next Database Directory Entry API 71

Get Next History File Entry API 129

Get Next Node Directory Entry API 458

Get Recommendations for a Health Indicator in Alert State API 109

Get Row Distribution Number API 493

Get Satellite Sync Session API 120

Get Snapshot API 113

Get SQLSTATE Message API 485

Get Table Space Statistics API 367

Get/Update Monitor Switches API 246

global pending list log record 725

GSS-APIs

- authentication plug-ins 667
- Restrictions 667

H

help

- displaying 761
- for SQL statements 760

heuristic abort log record 725

heuristic commit log record 725

host systems

- connections supported by DB2 Connect 436

I

identityignore file type modifier 139, 205

identitymissing file type modifier 139, 205

identityoverride file type modifier 205

implieddecimal file type modifier 139, 205

Import API 139

import replace (truncate) log record 725

import utility

- file type modifiers 153

importing

- code page considerations 139
- database access through DB2 Connect 139
- file to database table 139
- file type modifiers for 139
- PC/IXF, multiple-part files 139

- importing *(continued)*
 - restrictions 139
 - to a remote database 139
 - to a table or hierarchy that does not exist 139
 - to typed tables 139
- indexfreespace file type modifier 205
- indexixf file type modifier 139
- indexschema file type modifier 139
- Information Center
 - updating 762
 - versions 761
 - viewing in different languages 761
- INIT-INPUT structure 701
- INIT-OUTPUT structure 703
- Initialize and Link to Device API 680
- Initialize Read Log Without a Database Connection API 262
- initialize table log record 725
- Initialize the compression library API 708
- Initialize the decompression library API 714
- insert record log record 725
- Inspect database API 164
- Install Signal Handler API 453
- Instance Quiesce API 170
- Instance Start API 173
- Instance Stop API 179
- Instance Unquiesce API 183
- Interrupt API 451
- Interrupt Context API 450
- isolation levels
 - changing 755

K

- keepblanks file type modifier 139, 205
- Kerberos
 - security protocols
 - samples 671

L

- LDAP Deregister Server API 190
- LDAP Register Server API 192
- LDAP Update Alternate Server For Database API 203
- LDAP Update Server API 200
- libraries
 - security plug-in 614
 - restrictions on 620
- link file log record 725
- List DRDA Indoubt Transactions API 381
- List Indoubt Transactions API 345
- List SPM indoubt transactions API 322
- Load API 205
- load delete start compensation log record 725
- load pending list log record 725
- Load Query API 239
- load start log record 725
- load utility
 - file type modifiers 225
 - file type modifiers for 205
- lobsinfile
 - Export API 81
- lobsinfile file type modifier 139, 205
- local
 - database directory
 - open scan 75
- local pending list log record 725

- locks
 - changing 755
- log records
 - adding long field records 725
 - backout free 725
 - backup end 725
 - changing
 - table add columns 725
 - table attributes 725
 - creating
 - index 725
 - table 725
 - data manager 725
 - datalink manager 725
 - DB2 logs 725
 - deleting
 - groups 725
 - long field records 725
 - pgroups 725
 - records 725
 - DLFM prepare 725
 - dropping
 - index 725
 - tables 725
 - global pending list 725
 - headers 725
 - heuristic abort 725
 - heuristic commit 725
 - import replace (truncate) 725
 - initialize tables 725
 - insert records 725
 - link files 725
 - load delete start compensation 725
 - load pending list 725
 - load start 725
 - local pending list 725
 - long field manager 725
 - migration end 725
 - migration start 725
 - MPP coordinator commit 725
 - MPP subordinator commit 725
 - MPP subordinator prepare 725
 - non-update long field record 725
 - normal abort 725
 - normal commit 725
 - reorg table 725
 - rollback add columns 725
 - rollback create table 725
 - rollback delete record 725
 - rollback drop table 725
 - rollback insert 725
 - rollback update record 725
 - table load delete start 725
 - table space roll-forward to PIT begins 725
 - table space roll-forward to PIT ends 725
 - table space rolled forward 725
 - transaction manager 725
 - unlink file 725
 - update records 725
 - utility 725
 - XA prepare 725
- log sequence number (LSN) 725
- logs
 - recovery, allocating 408
- long field manager log records
 - add long field record 725
 - delete long field record 725

- long field manager log records (*continued*)
 - description 725
 - non-update long field record 725
- LSN (log sequence number) 725

M

- Migrate Database API 455
- migration
 - applications 751
- migration begin log record 725
- migration end log record 725
- modifiers
 - file type
 - export utility 89
 - import utility 153
 - load utility 225
- moving data
 - between databases 139
 - delimiter restrictions 237
- MPP coordinator commit log record 725
- MPP subordinator commit log record 725
- MPP subordinator prepare log record 725
- multiple concurrent requests
 - changing isolation levels 755

N

- nochecklengths file type modifier 139, 205
- nodefaults file type modifier 139
- nodes
 - directory 415
 - directory entries, retrieving 458
 - Open DCS Directory Scan API 445
 - SOCKS 546, 548
- nodoubledel file type modifier 81, 139, 205
- noeofchar file type modifier 139, 205
- noheader file type modifier 205
- nonpropagatable transactions 725
- nonupdate long field record log record 725
- normal abort log record 725
- normal commit log record 725
- norowwarnings file type modifier 205
- notices 765
- notypeid file type modifier 139
- nullindchar file type modifier 139, 205

O

- Open Database Directory Scan API 75
- Open DCS Directory Scan API 445
- Open History File Scan API 132
- Open Node Directory Scan API 460
- Open Table Space Container Query API 371
- Open Table Space Query API 373
- ordering DB2 books 760

P

- packages
 - creating 350
 - recreating 358
- packeddecimal file type modifier 205
- pagefreospace file type modifier 205

- partitions
 - obtaining table information 496
- passwords
 - changing with ATTACH 391
- performance
 - tuning
 - by reorganizing tables 271
- Ping Database API 60
- plug-ins
 - database management 591
 - group retrieval 627
 - GSS/API authentication 667
 - ID/password authentication 639
 - security
 - APIs 625
 - calling sequence, order plug-ins are called 617
 - deploying 606, 607, 609, 611
 - deployment limitations 613
 - error messages 625
 - library restrictions 620
 - names, naming conventions 598
 - restrictions on GSS-API 669
 - return codes 623
 - samples 671
 - versions of, versioning 670
- pointer manipulation 482, 483, 484
- Precompile Program API 355
- printed books
 - ordering 760
- privileges
 - database
 - granted when creating 408
- problem determination
 - online information 764
 - security plug-ins 604
 - tutorials 764
- propagatable transactions 725
- Prune History File API 249

Q

- Query Client API 462
- Query Client Information API 464
- Query Satellite Sync API 252
- Quiesce Table Spaces for Table API 498

R

- Read Log Without a Database Connection API 259
- Reading Data from Device API 685
- Rebind API 358
- reclen file type modifier
 - importing 139
 - Load API 205
- Recover Database API 265
- Redistribute Database Partition Group API 489
- redistributing data
 - in database partition groups 489
- reorg table log record 725
- Reorganize API 271
- Reset Alert Configuration API 279
- Reset Monitor API 281
- Restart Database API 64
- Restore database API 284
- return codes
 - description 21

- RETURN-CODE structure 705
- REXX language
 - API syntax 754
 - calling the DB2 CLP 754
- Roll Back an Indoubt Transaction API 504
- rollback add columns log record 725
- rollback create table log record 725
- rollback delete record log record 725
- rollback drop table log record 725
- rollback insert log record 725
- rollback update record log record 725
- Rollforward Database API 296
- Runstats API 306

S

- schemas
 - in new databases 408
- security
 - authentication 591
 - plug-ins
 - 32 bit considerations 603
 - 64 bit considerations 603
 - API for validating passwords 655
 - API versions 670
 - APIs 625, 630, 632, 633, 637, 638, 639, 646, 648, 649, 651, 653, 658, 659, 660, 661, 664, 665, 667
 - APIs for group retrieval 627
 - APIs for GSS-API 667
 - APIs for user ID/password 639
 - calling sequence of, order in which called 617
 - debugging, problem determination 604
 - deploying 606, 607, 609, 611
 - deployment 592, 613
 - developing 592
 - enabling 592
 - error messages 625
 - GSS-API 609
 - GSS-API on restrictions 669
 - initialization 614
 - libraries; location of security plug-in 597
 - limitations on deployment of plug-ins 613
 - loading 592, 614
 - naming 598
 - overview 592
 - restrictions on libraries 620
 - return codes 623
 - SQLCODES and SQLSTATES 604
 - two-part user ID support 600
 - XXXX 592
 - samples 671
 - user ID and password
 - database plug-ins 591
- Select the DB2 copy API 316
- Set Accounting String API 466
- Set Application Context Type API 476
- Set Client API 470
- Set Client Information API 473
- Set Configuration Parameters API 53
- Set Runtime Degree API 468
- Set Satellite Sync Session API 318
- Set Table Space Containers API 377
- signal handlers
 - Install Signal Handler API 453
 - Interrupt API 451
- Single Table Space Query API 375

- SOCKS node
 - using 546, 548
- SQL statements
 - displaying help 760
- sql_authorizations structure 513
- SQL-DIR-ENTRY structure 515
- sqlabndx API 350
- sqlaintp API 353
- sqlaprep API 355
- sqlarbnd API 358
- SQLB-TBS-STATS structure 517
- SQLB-TBSCONTQRY-DATA structure 519
- SQLB-TBSPQRY-DATA structure 521
- sqlbctcq API 361
- sqlbctsq API 362
- sqlbftcq API 363
- sqlbftpq API 365
- sqlbgtss API 367
- sqlbmtsq API 369
- sqlbotcq API 371
- sqlbotsq API 373
- sqlbstpq API 375
- sqlbstsc API 377
- sqlbtcq API 379
- SQLCA structure 526
 - retrieving error messages 21, 353, 485
- SQLCHAR structure 527
- SQLCODE values 21
- sqlcspqy API 381
- SQLDA structure 528
- SQLDB2 REXX API 754
- SQLDCOL structure 530
- sqlc_activate_db API 383
- sqlc_deactivate_db API 386
- SQLC-ADDN-OPTIONS structure 533
- SQLC-CLIENT-INFO structure 535
- SQLC-CONN-SETTING structure 538
- SQLC-NODE-CPIC structure 541
- SQLC-NODE-IPXSPX structure 542
- SQLC-NODE-LOCAL structure 543
- SQLC-NODE-NETB structure 544
- SQLC-NODE-NPIPE structure 545
- SQLC-NODE-STRUCT structure 546
- SQLC-NODE-TCPIP structure 548
- sqlcaddn API 388
- sqlcattcp API 391
- sqlcattin API 394
- sqlcAttachToCtx API 397
- sqlcBeginCtx API 398
- sqlccadb API 400
- sqlccran API 406
- sqlccrea API 408
- sqlcctnd API 415
- SQLCDBDESCEXT 556
- SQLCDBTERRITORYINFO structure 563
- sqlccgd API 419
- sqlcdcls API 69
- sqlcDetachFromCtx API 422
- sqlcdgne API 71
- sqlcdosd API 75
- sqlcdpan API 423
- sqlcdrpd API 425
- sqlcdrpn API 427
- sqlcdtin API 429
- sqlcEndCtx API 430
- sqlcfmem API 432
- sqlcfrcce API 433

- sqlgdad API 436
- sqlgdcl API 438
- sqlgdcl API 439
- sqlgdge API 441
- sqlgdgt API 443
- sqlgdsc API 445
- sqlGetCurrentCtx API 447
- sqlgins API 448
- sqlInterruptCtx API 450
- sqlintr API 451
- sqlleisg API 453
- sqlmgdb API 455
- sqlncls API 457
- sqlengne API 458
- SQLLENINFO structure 564
- sqlenops API 460
- sqlqryc API 462
- sqlqryi API 464
- sqlsact API 466
- sqlsdeg API 468
- sqlsetc API 470
- sqlseti API 473
- sqlSetTypeCtx API 476
- SQLETSDESC structure 549
- sqluncd API 478
- sqluncn API 480
- SQLFUPD structure 567
- sqlgaddr API 482
- sqlgdref API 483
- sqlgmcpy API 484
- sqllob structure 574
- SQLMA structure 575
- sqlogstt API 485
- SQLOPT structure 579
- SQLSTATE
 - messages 21
 - messages, retrieving from SQLSTATE field 485
- SQLU-LSN structure 581
- SQLU-MEDIA-LIST structure 582
- SQLU-RLOG-INFO structure 587
- sqludrdr API 489
- sqluexpr API 81
- sqlugrpn API 493
- sqlugtpi API 496
- sqluimpr API 139
- SQLUPI structure 588
- sqluvdel API 691
- sqluvend API 689
- sqluvget API 685
- sqluvint API 680
- sqluvput API 687
- sqluvqdp API 498
- SQLWARN messages 21
- SQLXA-XID structure 590
- sqlxhfrg API 501
- sqlxphcm API 502
- sqlxphrl API 504
- Start HADR API 121
- Stop HADR API 123
- Stop Satellite Sync API 326
- striptblanks file type modifier 139, 205
- striptnulls file type modifier 139, 205
- Sync Satellite API 325
- system database directory
 - cataloging 400
 - open scan 75
 - uncataloging 478

T

- Table Space Container Query API 379
- Table Space Query API 369
- table spaces
 - roll-forward to PIT begins log record 725
 - roll-forward to PIT ends log record 725
 - rolled forward log records 725
- tables
 - exporting to files 81
 - importing files 139
 - load delete start log record 725
- Take Over as Primary Database API 125
- TCP/IP
 - using SOCKS 546, 548
- Terminate Read Log Without a Database Connection API 264
- termination
 - abnormal 64
- terms and conditions
 - use of publications 765
- Test Satellite Sync API 327
- third party plug-ins 591
- timeformat file type modifier 139, 205
- timestampformat file type modifier 139, 205
- totalreespace file type modifier 205
- transaction identifier log records 725
- transaction managers
 - log records
 - backout free 725
 - description 725
 - global pending list 725
 - heuristic abort 725
 - heuristic commit 725
 - local pending list 725
 - MPP coordinator commit 725
 - MPP subordinator commit 725
 - MPP subordinator prepare 725
 - normal abort 725
 - normal commit 725
 - XA prepare 725
- troubleshooting
 - online information 764
 - security plug-ins 604
 - tutorials 764
- tutorials
 - troubleshooting and problem determination 764
 - Visual Explain 764

U

- Uncatalog Database API 478
- Uncatalog Database LDAP Entry API 196
- Uncatalog DCS Database API 439
- Uncatalog Node API 480
- Uncatalog Node LDAP Entry API 198
- uncataloging
 - system database directory 478
- unlink file log record 725
- Unlink the Device and Release its Resources API 689
- unsupported APIs and data structures 751
- Update Alert Configuration API 328
- update alternate server for database API 334
- Update Contact API 336
- Update Contact Group API 338
- Update Health Notification List API 340
- Update History File API 136
- update record log record 725

- updates
 - DB2 Information Center 762
 - Information Center 762
- usedefaults file type modifier 139, 205
- user IDs
 - two-part user IDs 600
- utility control API 342
- utility log records
 - backup end 725
 - description 725
 - load delete start compensation 725
 - load pending list 725
 - load start 725
 - migration begin 725
 - migration end 725
 - table load delete start 725
 - table space roll-forward to PIT begins 725
 - table space roll-forward to PIT ends 725
 - table space rolled forward 725

V

- vendor database plug-ins 591
- vendor products
 - backup and restore 671
 - DATA structure 704
 - description 671
 - INIT-INPUT structure 701
 - operation 671
- VENDOR-INFO structure 699
- Visual Explain
 - tutorial 764

W

- Writing Data to Device API 687

X

- XA prepare log record 725

Z

- zoned decimal file type modifier 205

Contacting IBM

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts at <http://www.ibm.com/planetwide>

To learn more about DB2 products, go to <http://www.ibm.com/software/data/db2/>.



Printed in USA

SC10-4231-00



Spine information:

IBM DB2 DB2 Version 9

Administrative API Reference

