

IBM DB2 10.5  
for Linux, UNIX, and Windows

*Administrative API Reference*





IBM DB2 10.5  
for Linux, UNIX, and Windows

*Administrative API Reference*



**Note**

Before using this information and the product it supports, read the general information under Appendix D, "Notices," on page 739.

**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 <http://www.ibm.com/shop/publications/order>
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at <http://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 IBM Corporation 1993, 2013.

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

How this book is structured. . . . . vii

Highlighting conventions . . . . . viii

## Chapter 1. DB2 APIs . . . . . 1

## Chapter 2. Changed APIs and data structures . . . . . 19

## Chapter 3. Log sequence number changes affecting API and application behavior . . . . . 23

## Chapter 4. How the API descriptions are organized . . . . . 27

Include files for DB2 API applications . . . . . 30

## Chapter 5. Administrative APIs . . . . . 33

db2AddContact - Add a contact to whom notification messages can be sent . . . . . 34

db2AddContactGroup - Add a contact group to whom notification messages can be sent. . . . . 36

db2AddSnapshotRequest - Add a snapshot request . . . . . 38

db2AdminMsgWrite - Write log messages for administration and replication function . . . . . 40

db2ArchiveLog - Archive the active log file. . . . . 42

db2AutoConfig - Access the Configuration Advisor . . . . . 44

db2AutoConfigFreeMemory - Free the memory allocated by the db2AutoConfig API . . . . . 48

db2Backup - Back up a database or table space . . . . . 49

db2CfgGet - Get the database manager or database configuration parameters . . . . . 59

db2CfgSet - Set the database manager or database configuration parameters . . . . . 62

db2ConvMonStream - Convert the monitor stream to the pre-version 6 format . . . . . 66

db2DatabasePing - Ping the database to test network response time . . . . . 69

db2DatabaseQuiesce - Quiesce the database . . . . . 71

db2DatabaseRestart - Restart database . . . . . 73

db2DatabaseUnquiesce - Unquiesce database . . . . . 76

db2DatabaseUpgrade - Upgrade previous version of DB2 database to the current release . . . . . 78

db2DbDirCloseScan - End a system or local database directory scan . . . . . 80

db2DbDirGetNextEntry - Get the next system or local database directory entry . . . . . 81

db2DbDirOpenScan - Start a system or local database directory scan . . . . . 85

db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent . . . . . 87

db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent . . . . . 88

db2Export - Export data from a database . . . . . 89

db2GetAlertCfg - Get the alert configuration settings for the health indicators . . . . . 96

db2GetAlertCfgFree - Free the memory allocated by the db2GetAlertCfg API. . . . . 101

db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent. . . . . 102

db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent. . . . . 104

db2GetContacts - Get the list of contacts to whom notification messages can be sent. . . . . 106

db2GetDistMap - Get distribution map. . . . . 108

db2GetHealthNotificationList - Get the list of contacts to whom health alert notifications can be sent . . . . . 110

db2GetRecommendations - Get recommendations to resolve a health indicator in alert state . . . . . 112

db2GetRecommendationsFree - Free the memory allocated by the db2GetRecommendations API . . . . . 115

db2GetSnapshot - Get a snapshot of the database manager operational status. . . . . 116

db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API. . . . . 120

db2GetSyncSession - Get a satellite synchronization session identifier . . . . . 123

db2HADRStart - Start high availability disaster recovery (HADR) operations . . . . . 124

db2HADRStop - Stop high availability disaster recovery (HADR) operations . . . . . 126

db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database . . . . . 128

db2HistoryCloseScan - End the database history records scan. . . . . 131

db2HistoryGetEntry - Get the next entry in the database history records. . . . . 132

db2HistoryOpenScan - Start a database history records scan. . . . . 134

db2HistoryUpdate - Update a database history records entry . . . . . 138

db2Import - Import data into a table, hierarchy, nickname or view . . . . . 141

db2Ingest API- Ingest data from an input file or pipe into a DB2 table. . . . . 154

db2Inspect - Inspect database for architectural integrity . . . . . 166

db2InstanceQuiesce - Quiesce instance . . . . . 173

db2InstanceStart - Start instance . . . . . 176

db2InstanceStop - Stop instance . . . . . 182

db2InstanceUnquiesce - Unquiesce instance . . . . . 186

db2LdapCatalogDatabase - Register the database on the LDAP server . . . . . 188

db2LdapCatalogNode - Provide an alias for node name in LDAP server . . . . .	190
db2LdapDeregister - Deregister the DB2 server and cataloged databases from the LDAP server . . .	191
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. . . . .	197
db2LdapUpdate - Update the attributes of the DB2 server on the LDAP server . . . . .	198
db2LdapUpdateAlternateServerForDB - Update the alternate server for the database on the LDAP server . . . . .	201
db2Load - Load data into a table. . . . .	203
db2LoadQuery - Get the status of a load operation	224
db2MonitorSwitches - Get or update the monitor switch settings . . . . .	231
db2Prune - Delete the history file entries or log files from the active log path . . . . .	234
db2QuerySatelliteProgress - Get the status of a satellite synchronization session . . . . .	237
db2ReadLog - Read log records . . . . .	239
db2ReadLogNoConn - Read the database logs without a database connection. . . . .	245
db2ReadLogNoConnInit - Initialize reading the database logs without a database connection . .	248
db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection . .	250
db2Recover - Restore and roll forward a database	251
db2Reorg - Reorganize an index or a table. . . .	257
db2ResetAlertCfg - Reset the alert configuration of health indicators . . . . .	267
db2ResetMonitor - Reset the database system monitor data . . . . .	269
db2Restore - Restore a database or table space .	272
db2Rollforward - Roll forward a database. . . .	286
db2Runstats - Update statistics for tables and indexes . . . . .	296
db2SelectDB2Copy - Select the DB2 copy to be used by your application . . . . .	307
db2SetSyncSession - Set satellite synchronization session . . . . .	309
db2SetWriteForDB - Suspend or resume I/O writes for database. . . . .	310
db2SpmListIndTrans - List SPM indoubt transactions . . . . .	312
db2SyncSatellite - Start satellite synchronization	315
db2SyncSatelliteStop - Pause satellite synchronization . . . . .	316
db2SyncSatelliteTest - Test whether a satellite can be synchronized . . . . .	317
db2UpdateAlertCfg - Update the alert configuration settings for health indicators . .	318
db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory. . . . .	324
db2UpdateContact - Update the attributes of a contact . . . . .	326

db2UpdateContactGroup - Update the attributes of a contact group. . . . .	328
db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent . . . . .	330
db2UtilityControl - Set the priority level of running utilities . . . . .	332
sqlabndx - Bind application program to create a package . . . . .	334
sqlaintp - Get error message . . . . .	337
sqlaprep - Precompile application program . . .	339
sqlarbnd - Rebind package . . . . .	342
sqlbctcq - Close a table space container query .	345
sqlbctsq - Close a table space query . . . . .	346
sqlbftcq - Fetch the query data for rows in a table space container. . . . .	347
sqlbftpq - Fetch the query data for rows in a table space . . . . .	349
sqlbgts - Get table space usage statistics . . .	351
sqlbmstsq - Get the query data for all table spaces	353
sqlbotcq - Open a table space container query .	355
sqlbotsq - Open a table space query . . . . .	357
sqlbstpq - Get information about a single table space . . . . .	359
sqlbstsc - Set table space containers . . . . .	361
sqlbtcq - Get the query data for all table space containers . . . . .	364
sqlcspqy - List DRDA indoubt transactions . . .	366
sqle_activate_db - Activate database. . . . .	367
sqle_deactivate_db - Deactivate database . . . .	370
sqleaddn - Add a database partition to the partitioned database environment . . . . .	373
sqleatcp - Attach to instance and change password	375
sqleatin - Attach to instance . . . . .	378
sqlecadb - Catalog a database in the system database directory. . . . .	381
sqlecran - Create a database on a database partition server . . . . .	387
sqlecrea - Create database . . . . .	389
sqlectnd - Catalog an entry in the node directory	397
sqledcgd - Change a database comment in the system or local database directory . . . . .	400
sqledpan - Drop a database on a database partition server . . . . .	402
sqledrpd - Drop database . . . . .	404
sqledrpn - Check whether a database partition server can be dropped . . . . .	406
sqledtin - Detach from instance . . . . .	408
sqlefmem - Free the memory allocated by the sqlbtcq and sqlbmstsq API . . . . .	409
sqlefrce - Force users and applications off the system . . . . .	410
sqlegdad - Catalog a database in the database connection services (DCS) directory . . . . .	413
sqlegdcl - End a database connection services (DCS) directory scan . . . . .	415
sqlegdel - Uncatalog a database from the database connection services (DCS) directory . . . . .	416
sqlegdge - Get a specific entry in the database connection services (DCS) directory . . . . .	418

sqllegdgt - Get database connection services (DCS) directory entries . . . . .	420
sqlgdsc - Start a database connection services (DCS) directory scan . . . . .	422
sqlgins - Get current instance. . . . .	423
sqlintr - Interrupt application requests. . . . .	424
sqlsig - Install signal handler . . . . .	426
sqlmgdb - Upgrade previous version of DB2 database to current version. . . . .	427
sqlncsl - End a node directory scan. . . . .	429
sqlngne - Get the next node directory entry . . . . .	430
sqlnops - Start a node directory scan . . . . .	432
sqlqryc - Query client connection settings . . . . .	434
sqlqryi - Query client information . . . . .	436
sqlsact - Set accounting string . . . . .	438
sqlsdeg - Set the maximum runtime intrapartition parallelism level or degree for SQL statements . . . . .	439
sqlsetc - Set client connection settings . . . . .	441
sqlseti - Set client information . . . . .	444
sqluncd - Uncatalog a database from the system database directory. . . . .	446
sqluncn - Uncatalog an entry from the node directory . . . . .	448
sqlgaddr - Get the address of a variable . . . . .	450
sqlgdref - Dereference an address. . . . .	451
sqlgmcpy - Copy data from one memory area to another . . . . .	452
sqlgstt - Get the SQLSTATE message . . . . .	453
sqludrtd - Redistribute data across a database partition group. . . . .	455
sqlugrpn - Get the database partition server number for a row . . . . .	459
sqlugtpi - Get table distribution information . . . . .	462
sqluvqdp - Quiesce table spaces for a table . . . . .	464

<b>Chapter 6. Calling DB2 APIs in REXX</b>	<b>467</b>
Change Isolation Level . . . . .	469

<b>Chapter 7. Indoubt transaction management APIs</b>	<b>471</b>
db2XaGetInfo - Get information for a resource manager . . . . .	473
db2XaListIndTrans - List indoubt transactions . . . . .	474
sqlxhfrg - Forget transaction status . . . . .	479
sqlxphcm - Commit an indoubt transaction . . . . .	480
sqlxphrl - Roll back an indoubt transaction . . . . .	481

<b>Chapter 8. Threaded applications with concurrent access</b>	<b>483</b>
sqlAttachToCtx - Attach to context . . . . .	484
sqlBeginCtx - Create and attach to an application context . . . . .	485
sqlDetachFromCtx - Detach from context. . . . .	486
sqlEndCtx - Detach from and free the memory associated with an application context . . . . .	487
sqlGetCurrentCtx - Get current context . . . . .	489
sqlInterruptCtx - Interrupt context . . . . .	490
sqlSetTypeCtx - Set application context type. . . . .	491

<b>Chapter 9. DB2 database system plug-ins for customizing database management.</b>	<b>493</b>
Enabling plug-ins . . . . .	493
Deploying a group retrieval plug-in . . . . .	493
Deploying a user ID/password plug-in. . . . .	494
Deploying a GSS-API plug-in . . . . .	495
Deploying a Kerberos plug-in . . . . .	496
Writing security plug-ins . . . . .	498
How DB2 loads security plug-ins. . . . .	498
Restrictions for developing security plug-in libraries . . . . .	500
Restrictions on security plug-ins . . . . .	502
Return codes for security plug-ins . . . . .	504
Error message handling for security plug-ins . . . . .	507
Calling sequences for the security plug-in APIs . . . . .	508
Security plug-ins . . . . .	510
Security plug-in library locations . . . . .	515
Security plug-in naming conventions . . . . .	516
Security plug-in support for two-part user IDs . . . . .	517
Security plug-in API versioning . . . . .	518
32-bit and 64-bit considerations for security plug-ins . . . . .	519
Security plug-in problem determination . . . . .	519
Security plug-in APIs. . . . .	520
APIs for group retrieval plug-ins . . . . .	522
db2secDoesGroupExist API - Check if group exists . . . . .	524
db2secFreeErrorMsg API - Free error message memory . . . . .	525
db2secFreeGroupListMemory API - Free group list memory . . . . .	526
db2secGetGroupsForUser API - Get list of groups for user. . . . .	527
db2secGroupPluginInit API - Initialize group plug-in . . . . .	530
db2secPluginTerm - Clean up group plug-in resources . . . . .	532
APIs for user ID/password authentication plug-ins . . . . .	533
db2secClientAuthPluginInit API - Initialize client authentication plug-in . . . . .	539
db2secClientAuthPluginTerm API - Clean up client authentication plug-in resources . . . . .	541
db2secDoesAuthIDExist - Check if authentication ID exists . . . . .	542
db2secFreeInitInfo API - Clean up resources held by the db2secGenerateInitialCred . . . . .	543
db2secFreeToken API - Free memory held by token . . . . .	544
db2secGenerateInitialCred API - Generate initial credentials . . . . .	545
db2secGetAuthIDs API - Get authentication IDs . . . . .	547
db2secGetDefaultLoginContext API - Get default login context . . . . .	549
db2secProcessServerPrincipalName API - Process service principal name returned from server . . . . .	551
db2secRemapUserid API - Remap user ID and password. . . . .	552



db2secServerAuthPluginInit - Initialize server authentication plug-in . . . . .	554
db2secServerAuthPluginTerm API - Clean up server authentication plug-in resources . . . . .	557
db2secValidatePassword API - Validate password. . . . .	558
Required APIs and definitions for GSS-API authentication plug-ins . . . . .	561
Restrictions for GSS-API authentication plug-ins	563
Security plug-in samples . . . . .	564
DB2 APIs for backup and restore to storage managers. . . . .	565
db2VendorGetNextObj - Get next object on device. . . . .	566
db2VendorQueryApiVersion - Get the supported level of the vendor storage API . . . . .	568
sqluvdel - Delete committed session. . . . .	569
sqluvend - Unlink a vendor device and release its resources. . . . .	570
sqluvget - Read data from a vendor device . . . . .	572
sqluvint - Initialize and link to a vendor device	574
sqluvput - Write data to a vendor device . . . . .	578
DB2_info . . . . .	580
Vendor_info . . . . .	583
Init_input . . . . .	585
Init_output . . . . .	587
Data . . . . .	588
Return_code. . . . .	589
DB2 APIs for using compression with backup and restore operations . . . . .	590
COMPR_CB . . . . .	592
COMPR_DB2INFO . . . . .	593
COMPR_PIINFO . . . . .	595
Compress - Compress a block of data . . . . .	597
Decompress - Decompress a block of data. . . . .	599
GetMaxCompressedSize - Estimate largest possible buffer size . . . . .	600
GetSavedBlock - Get the vendor of block data for the backup image. . . . .	601
InitCompression - Initialize the compression library. . . . .	602
InitDecompression - Initialize the decompression library . . . . .	603
TermCompression - Stop the compression library. . . . .	604
TermDecompression - Stop the decompression library. . . . .	605

## Chapter 10. Data structures used by APIs . . . . . 607

db2DistMapStruct . . . . .	608
db2HistoryData . . . . .	609
db2LSN data structure . . . . .	614
sql_dir_entry . . . . .	615
SQLB_TBS_STATS. . . . .	616

SQLB_TBSCONTQRY_DATA . . . . .	617
SQLB_TBSPQRY_DATA . . . . .	619
SQLCA . . . . .	623
sqlchar . . . . .	624
SQLDA . . . . .	625
sqldcol . . . . .	627
sqle_addn_options. . . . .	630
sqle_client_info. . . . .	632
sqle_conn_setting . . . . .	634
sqle_node_local. . . . .	637
sqle_node_npip . . . . .	638
sqle_node_struct . . . . .	639
sqle_node_tcpip . . . . .	640
sqledbdesc . . . . .	641
sqledbdescext . . . . .	649
sqledbterritoryinfo. . . . .	655
sqleninfo . . . . .	656
sqlfupd . . . . .	659
sqllob . . . . .	668
sqlma . . . . .	669
sqlopt . . . . .	672
SQLU_LSN . . . . .	674
sqlu_media_list. . . . .	675
SQLU_RLOG_INFO . . . . .	680
sqlupi . . . . .	681
SQLXA_XID. . . . .	683

## Appendix A. Precompiler customization APIs . . . . . 685

Precompiler customization APIs . . . . .	686
--	-----

## Appendix B. DB2 log records . . . . . 687

DB2 log records . . . . .	688
Log record header. . . . .	690
Transaction manager log records . . . . .	692
Long field manager log records . . . . .	699
Utility manager log records. . . . .	701
Data manager log records . . . . .	704
Relation manager log records . . . . .	722
LOB manager log records . . . . .	728

## Appendix C. Overview of the DB2 technical information . . . . . 733

DB2 technical library in hardcopy or PDF format	734
Displaying SQL state help from the command line processor. . . . .	736
Accessing different versions of the DB2 Information Center . . . . .	736
Terms and conditions. . . . .	736

## Appendix D. Notices . . . . . 739

## Index . . . . . 743



---

## About this book

This book provides information about the use of application programming interfaces (APIs) to execute database administrative functions. It presents detailed information about 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, and/or REXX programming languages
- Application program design

---

## How this book is structured

This book provides the reference information needed to use the administrative APIs in application development.

The major subject areas discussed in the chapters of this book are as follows:

### Overview of administrative APIs and data structures

- Chapter 1, “DB2<sup>®</sup> APIs,” includes tables that list administrative APIs, include files, and sample programs.
- Chapter 2, “Changed APIs and data structures,” uses tables to list supported and unsupported APIs and data structures that have changed.
- Chapter 3, “How the API descriptions are organized,” describes how API descriptions are organized and lists the include files for DB2 API applications.

### APIs

- Chapter 4, “Administrative APIs,” alphabetically lists the DB2 administrative APIs.
- Chapter 5, “Calling DB2 APIs in REXX,” describes how to call DB2 APIs from a REXX application.
- Chapter 6, “Indoubt transaction management APIs,” presents a set of APIs provided for tool writers to perform heuristic functions on indoubt transactions.
- Chapter 7, “Threaded applications with concurrent access,” describes DB2 APIs that can be used in threaded applications.

## How this book is structured

### Plug-in APIs

- Chapter 8, “DB2 database system plug-ins for customizing database management,” presents the security, backup, restore, log archiving, and compression/decompression for backup images plug-in APIs that you and third-party vendors can use to customize certain database management functions.

### Data structures

- Chapter 9, “Data structures used by APIs,” describes the data structures used by APIs.

### Appendixes

- Appendix A, “Precompiler customization APIs,” provides a link on where to obtain information about a set of documented APIs that enable other application development tools to implement precompiler support for DB2 directly within their products.
- Appendix B, “DB2 log records,” describes the structure of the various DB2 log records.

---

## Highlighting conventions

The following highlighting conventions are used in this book.

<b>Bold</b>	Indicates commands, keywords, and other items whose names are predefined by the system.
<i>Italics</i>	Indicates one of the following values: <ul style="list-style-type: none"><li>• Names or values (variables) that must be supplied by the user</li><li>• General emphasis</li><li>• The introduction of a new term</li><li>• A reference to another source of information</li></ul>
Monospace	Indicates one of the following values: <ul style="list-style-type: none"><li>• Files and directories</li><li>• Information that you are instructed to type at a command prompt or in a window</li><li>• Examples of specific data values</li><li>• Examples of text similar to what might be displayed by the system</li><li>• Examples of system messages</li><li>• Samples of programming code</li></ul>

---

## Chapter 1. DB2 APIs

Lists the DB2 APIs that are demonstrated in the DB2 samples.

**Important:** DB2 Administrative APIs are not supported in multi-threaded applications, unless you explicitly manage the serialization of threads with the use of DB2 context APIs. Instead of DB2 Administrative APIs along with DB2 context APIs, use the SYSPROC.ADMIN\_CMD stored procedure and rely on the DB2 database manager to automatically manage the contexts for you with its multi-threaded application support. For details about multi-threaded application support and using contexts for threaded applications with concurrent access, see: “Concurrent transactions and multi-threaded database access in embedded SQL applications” in *Developing Embedded SQL Applications*.

The first table lists the DB2 APIs grouped by functional category, their corresponding include files, and the sample programs that demonstrate them (see the note after the table for more information about 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 13.

### COBOL Sample Programs with DB2 APIs

Table 3 on page 16.

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

API Type	DB2 API	Include File	Sample Programs
Database control APIs	“db2DatabaseQuiesce - Quiesce the database” on page 71	db2ApiDf	n/a
Database control APIs	“db2DatabaseUnquiesce - Unquiesce database” on page 76	db2ApiDf	n/a
Database control APIs	“db2DatabaseRestart - Restart database” on page 73	db2ApiDf	C: dbconn.sqc C++: dbconn.sqC
Database control APIs	“sqlcrea - Create database” on page 389	sqlenv	C: dbcreate.c dbrecov.sqc dbsample.sqc C++: dbcreate.C dbrecov.sq COBOL: db_udcs.cbl dbconf.cbl ebcidcdb.cbl
Database control APIs	“sqlcran - Create a database on a database partition server” on page 387	sqlenv	n/a
Database control APIs	“sqldrpd - Drop database” on page 404	sqlenv	C: dbcreate.c C++: dbcreate.C COBOL: dbconf.cbl
Database control APIs	“sqldpan - Drop a database on a database partition server” on page 402	sqlenv	n/a
Database control APIs	“db2DatabaseUpgrade - Upgrade previous version of DB2 database to the current release” on page 78	db2ApiDf	C: dbupgrade.c C++: dbupgrade.C COBOL: dbupgrade.cbl
Database control APIs	“db2XaListIndTrans - List indoubt transactions” on page 474	db2ApiDf	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	"sqlc_activate_db - Activate database" on page 367	sqlenv	n/a
Database control APIs	"sqlc_deactivate_db - Deactivate database" on page 370	sqlenv	n/a
Database control APIs	"sqlcspqy - List DRDA indoubt transactions" on page 366	sqlxa	n/a
Database control APIs	"db2SetWriteForDB - Suspend or resume I/O writes for database" on page 310	db2ApiDf	n/a
Database control APIs	"sqlcfrce - Force users and applications off the system" on page 410	sqlenv	C: dbconn.sqc dbsample.sqc instart.c C++: dbconn.sqC instart.C COBOL: dbstop.cb1
Instance control APIs	"db2InstanceStart - Start instance" on page 176	db2ApiDf	C: instart.c C++: instart.C
Instance control APIs	"db2InstanceStop - Stop instance" on page 182	db2ApiDf	C: instart.c C++: instart.C
Instance control APIs	"db2InstanceQuiesce - Quiesce instance" on page 173	db2ApiDf	n/a
Instance control APIs	"db2InstanceUnquiesce - Unquiesce instance" on page 186	db2ApiDf	n/a
Instance control APIs	"sqlcattach - Attach to instance" on page 378	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cb1
Instance control APIs	"sqlcatcp - Attach to instance and change password" on page 375	sqlenv	C: inattach.c C++: inattach.C COBOL: dbinst.cb1
Instance control APIs	"sqledtin - Detach from instance" on page 408	sqlenv	C: inattach.c utilapi.c C++: inattach.C utilapi.C COBOL: dbinst.cb1
Instance control APIs	"sqlcgetins - Get current instance" on page 423	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbinst.cb1
Instance control APIs	"db2UtilityControl - Set the priority level of running utilities" on page 332	db2ApiDf	n/a
Database manager and database configuration APIs	"db2CfGet - Get the database manager or database configuration parameters" on page 59	db2ApiDf	C: dbinfo.c dbrecov.sqc ininfo.c tscreate.sqc C++: dbinfo.C dbrecov.sqC ininfo.C tscreate.sqC
Database manager and database configuration APIs	"db2CfSet - Set the database manager or database configuration parameters" on page 62	db2ApiDf	C: dbinfo.c dbrecov.sqc ininfo.c C++: dbinfo.C dbrecov.sqC ininfo.C
Database manager and database configuration APIs	"db2AutoConfig - Access the Configuration Advisor" on page 44	db2AuCfg	C: dbcfg.sqc C++: dbcfg.sqC

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

API Type	DB2 API	Include File	Sample Programs
Database manager and database configuration APIs	"db2AutoConfigFreeMemory - Free the memory allocated by the db2AutoConfig API" on page 48	db2AuCfgr	C: dbcfg.sqc C++: dbcfg.sqc
Database monitoring APIs	"db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API" on page 120	db2ApiDf	n/a
Database monitoring APIs	"db2AddSnapshotRequest - Add a snapshot request" on page 38	db2ApiDf	n/a
Database monitoring APIs	"db2MonitorSwitches - Get or update the monitor switch settings" on page 231	db2ApiDf	C: utilsnap.c C++: utilsnap.C
Database monitoring APIs	"db2GetSnapshot - Get a snapshot of the database manager operational status" on page 116	db2ApiDf	C: utilsnap.c C++: utilsnap.C
Database monitoring APIs	"db2ResetMonitor - Reset the database system monitor data" on page 269	db2ApiDf	n/a
Database monitoring APIs	"db2ConvMonStream - Convert the monitor stream to the pre-version 6 format" on page 66	db2ApiDf	n/a
Database monitoring APIs	"db2Inspect - Inspect database for architectural integrity" on page 166	db2ApiDf	n/a
Database health monitoring APIs	"db2AddContact - Add a contact to whom notification messages can be sent" on page 34	db2ApiDf	n/a
Database health monitoring APIs	"db2AddContactGroup - Add a contact group to whom notification messages can be sent" on page 36	db2ApiDf	n/a
Database health monitoring APIs	"db2DropContact - Remove a contact from the list of contacts to whom notification messages can be sent" on page 87	db2ApiDf	n/a
Database health monitoring APIs	"db2DropContactGroup - Remove a contact group from the list of contacts to whom notification messages can be sent" on page 88	db2ApiDf	n/a
Database health monitoring APIs	"db2GetAlertCfg - Get the alert configuration settings for the health indicators" on page 96	db2ApiDf	n/a
Database health monitoring APIs	"db2GetAlertCfgFree - Free the memory allocated by the db2GetAlertCfg API" on page 101	db2ApiDf	n/a
Database health monitoring APIs	"db2GetContactGroup - Get the list of contacts in a single contact group to whom notification messages can be sent" on page 102	db2ApiDf	n/a
Database health monitoring APIs	"db2GetContactGroups - Get the list of contact groups to whom notification messages can be sent" on page 104	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	"db2GetContacts - Get the list of contacts to whom notification messages can be sent" on page 106	db2ApiDf	n/a
Database health monitoring APIs	"db2GetHealthNotificationList - Get the list of contacts to whom health alert notifications can be sent" on page 110	db2ApiDf	n/a
Database health monitoring APIs	"db2ResetAlertCfg - Reset the alert configuration of health indicators" on page 267	db2ApiDf	n/a
Database health monitoring APIs	"db2UpdateAlertCfg - Update the alert configuration settings for health indicators" on page 318	db2ApiDf	n/a
Database health monitoring APIs	"db2UpdateContact - Update the attributes of a contact" on page 326	db2ApiDf	n/a
Database health monitoring APIs	"db2UpdateContactGroup - Update the attributes of a contact group" on page 328	db2ApiDf	n/a
Database health monitoring APIs	"db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent" on page 330	db2ApiDf	n/a
Database health monitoring APIs	"db2GetSnapshot - Get a snapshot of the database manager operational status" on page 116	db2ApiDf	C: utilsnap.c C++: utilsnap.C
Database health monitoring APIs	"db2GetSnapshotSize - Estimate the output buffer size required for the db2GetSnapshot API" on page 120	db2ApiDf	n/a
Database health monitoring APIs	"db2GetRecommendations - Get recommendations to resolve a health indicator in alert state" on page 112	db2ApiDf	n/a
Database health monitoring APIs	"db2GetRecommendationsFree - Free the memory allocated by the db2GetRecommendations API" on page 115	db2ApiDf	n/a
Data movement APIs	"db2Export - Export data from a database" on page 89	sqlutil	C: tbmove.sqc C++: tbmove.sqc COBOL: expsamp.sqb impexp.sqb tload.sqb
Data movement APIs	"db2Import - Import data into a table, hierarchy, nickname or view" on page 141	db2ApiDf	C: dtformat.sqc tbmove.sqc C++: tbmove.sqc COBOL: expsamp.sqb impexp.sqb
Data movement APIs	"db2Load - Load data into a table" on page 203	db2ApiDf	C: dtformat.sqc tload.sqc tbmove.sqc C++: tbmove.sqc
Data movement APIs	"db2LoadQuery - Get the status of a load operation" on page 224	db2ApiDf	C: tbmove.sqc C++: tbmove.sqc COBOL: loadqry.sqb
Data movement APIs	db2Ingest API- Ingest data from an input file or pipe into a DB2 table	db2ApiDf	n/a
Recovery APIs	"db2Backup - Back up a database or table space" on page 49	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2Restore - Restore a database or table space" on page 272	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc

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

API Type	DB2 API	Include File	Sample Programs
Recovery APIs	"db2Recover - Restore and roll forward a database" on page 251	db2ApiDf	n/a
Recovery APIs	"db2Rollforward - Roll forward a database" on page 286	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2HistoryOpenScan - Start a database history records scan" on page 134	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2HistoryGetEntry - Get the next entry in the database history records" on page 132	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2HistoryCloseScan - End the database history records scan" on page 131	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2Prune - Delete the history file entries or log files from the active log path" on page 234	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2HistoryUpdate - Update a database history records entry" on page 138	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqc
Recovery APIs	"db2ArchiveLog - Archive the active log file" on page 42	db2ApiDf	n/a
High Availability Disaster Recovery (HADR) APIs	"db2HADRStart - Start high availability disaster recovery (HADR) operations" on page 124	db2ApiDf	n/a
High Availability Disaster Recovery (HADR) APIs	"db2HADRStop - Stop high availability disaster recovery (HADR) operations" on page 126	db2ApiDf	n/a
High Availability Disaster Recovery (HADR) APIs	"db2HADRTakeover - Instruct a database to take over as the high availability disaster recovery (HADR) primary database" on page 128	db2ApiDf	n/a
Database directory and DCS directory management APIs	"sqlcadb - Catalog a database in the system database directory" on page 381	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl
Database directory and DCS directory management APIs	"sqlcuncd - Uncatalog a database from the system database directory" on page 446	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl
Database directory and DCS directory management APIs	"sqlcgdad - Catalog a database in the database connection services (DCS) directory" on page 413	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl



## 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	"sqlgdel - Uncatalog a database from the database connection services (DCS) directory" on page 416	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
Database directory and DCS directory management APIs	"sqledcgl - Change a database comment in the system or local database directory" on page 400	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dbcmt.cbl
Database directory and DCS directory management APIs	"db2DbDirOpenScan - Start a system or local database directory scan" on page 85	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
Database directory and DCS directory management APIs	"db2DbDirGetNextEntry - Get the next system or local database directory entry" on page 81	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
Database directory and DCS directory management APIs	"db2DbDirCloseScan - End a system or local database directory scan" on page 80	db2ApiDf	C: ininfo.c C++: ininfo.C COBOL: dbcat.cbl dbcmt.cbl
Database directory and DCS directory management APIs	"sqlgdesc - Start a database connection services (DCS) directory scan" on page 422	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
Database directory and DCS directory management APIs	"sqlgdgt - Get database connection services (DCS) directory entries" on page 420	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
Database directory and DCS directory management APIs	"sqlgdcl - End a database connection services (DCS) directory scan" on page 415	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
Database directory and DCS directory management APIs	"sqlgdge - Get a specific entry in the database connection services (DCS) directory" on page 418	sqlenv	C: ininfo.c C++: ininfo.C COBOL: dcscat.cbl
Database directory and DCS directory management APIs	"db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory" on page 324	db2ApiDf	n/a
Client/server management APIs	"sqlqryc - Query client connection settings" on page 434	sqlenv	C: cli_info.c C++: cli_info.C COBOL: client.cbl

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

API Type	DB2 API	Include File	Sample Programs
Client/server management APIs	"sqlqryi - Query client information" on page 436	sqlenv	C: cli_info.c C++: cli_info.C
Client/server management APIs	"sqlsetc - Set client connection settings" on page 441	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 - Set client information" on page 444	sqlenv	C: cli_info.c C++: cli_info.C
Client/server management APIs	"sqlsact - Set accounting string" on page 438	sqlenv	COBOL: setact.cbl
Client/server management APIs	"db2DatabasePing - Ping the database to test network response time" on page 69	db2ApiDf	n/a
Client/server management APIs	"sqlsig - Install signal handler" on page 426	sqlenv	COBOL: dbcmt.cbl
Client/server management APIs	"sqlintr - Interrupt application requests" on page 424	sqlenv	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapRegister - Register the DB2 server on the LDAP server" on page 192	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapUpdate - Update the attributes of the DB2 server on the LDAP server" on page 198	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapDeregister - Deregister the DB2 server and cataloged databases from the LDAP server" on page 191	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapCatalogNode - Provide an alias for node name in LDAP server" on page 190	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapUncatalogNode - Delete alias for node name from LDAP server" on page 197	db2ApiDf	n/a

## DB2 APIs

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	"db2LdapCatalogDatabase - Register the database on the LDAP server" on page 188	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapUncatalogDatabase - Deregister database from LDAP server" on page 196	db2ApiDf	n/a
Lightweight Directory Access Protocol (LDAP) directory management APIs	"db2LdapUpdateAlternateServerForDB - Update the alternate server for the database on the LDAP server" on page 201	db2ApiDf	n/a
Application programming and preparation APIs	"sqlaintp - Get error message" on page 337	sql	C: dbcfg.sqcutilapi.c C++: dbcfg.sqlcutilapi.C COBOL: checkerr.cbl
Application programming and preparation APIs	"sqllogstt - Get the SQLSTATE message" on page 453	sql	C: utilapi.c C++: utilapi.C COBOL: checkerr.cbl
Application programming and preparation APIs	"sqleisig - Install signal handler" on page 426	sqlenv	COBOL: dbcmt.cbl
Application programming and preparation APIs	"sqlintr - Interrupt application requests" on page 424	sqlenv	n/a
Application programming and preparation APIs	"sqlaprep - Precompile application program" on page 339	sql	C: dbpkg.sqc C++: dbpkg.sqlC
Application programming and preparation APIs	"sqlabndx - Bind application program to create a package" on page 334	sql	C: dbpkg.sqc dbsample.sqc C++: dbpkg.sqlC
Application programming and preparation APIs	"sqlarbind - Rebind package" on page 342	sql	C: dbpkg.sqc C++: dbpkg.sqlC COBOL: rebind.sqb
COBOL, FORTRAN and REXX application specific APIs	"sqlgaddr - Get the address of a variable" on page 450	sqlutil	n/a

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

API Type	DB2 API	Include File	Sample Programs
COBOL, FORTRAN and REXX application specific APIs	"sqlgdref - Dereference an address" on page 451	sqlutil	n/a
COBOL, FORTRAN and REXX application specific APIs	"sqlgmcpy - Copy data from one memory area to another" on page 452	sqlutil	n/a
Table space and table management APIs	"sqlbtcq - Get the query data for all table space containers" on page 364 This command or API has been deprecated and might be removed in a future release. See tsinfo.db2 tsinfo.db2 for a sample program that uses a replacement function.	sqlutil	n/a
Table space and table management APIs	"sqlbotcq - Open a table space container query" on page 355	sqlutil	COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	"sqlbftcq - Fetch the query data for rows in a table space container" on page 347	sqlutil	COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	"sqlbctcq - Close a table space container query" on page 345	sqlutil	COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	"sqlbstsc - Set table space containers" on page 361	sqlutil	C: dbrecov.sqc C++: dbrecov.sqC COBOL: tabscont.sqb tspace.sqb
Table space and table management APIs	"sqlbmtsq - Get the query data for all table spaces" on page 353 This command or API has been deprecated and might be removed in a future release. See tsinfo.db2 tsinfo.db2 for a sample program that uses a replacement function.	sqlutil	n/a
Table space and table management APIs	"sqlbstpq - Get information about a single table space" on page 359	sqlutil	COBOL: tabspace.sqb tspace.sqb
Table space and table management APIs	"sqlbotsq - Open a table space query" on page 357 This command or API has been deprecated and might be removed in a future release.	sqlutil	n/a
Table space and table management APIs	"sqlbftpq - Fetch the query data for rows in a table space" on page 349	sqlutil	COBOL: tabspace.sqb tspace.sqb

## DB2 APIs

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

API Type	DB2 API	Include File	Sample Programs
Table space and table management APIs	"sqlbctsq - Close a table space query" on page 346 This command or API has been deprecated and might be removed in a future release.	sqlutil	n/a
Table space and table management APIs	This command or API has been deprecated and might be removed in a future release.	sqlutil	n/a
Table space and table management APIs	"sqluvqdp - Quiesce table spaces for a table" on page 464	sqlutil	C: tbmove.sqc C++: tbmove.sqC COBOL: tload.sqb
Table space and table management APIs	"db2Runstats - Update statistics for tables and indexes" on page 296	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
Table space and table management APIs	"db2Reorg - Reorganize an index or a table" on page 257	db2ApiDf	C: tbreorg.sqc C++: tbreorg.sqC COBOL: dbstat.sqb
Table space and table management APIs	"sqlfmem - Free the memory allocated by the sqlbtcq and sqlbmtsq API" on page 409	sqlenv	C: dbrecov.sqc C++: dbrecov.sqC COBOL: tabscont.sqb tabspace.sqb tspace.sqb
Node directory management APIs	"sqlctnd - Catalog an entry in the node directory" on page 397	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
Node directory management APIs	"sqlleun - Uncatalog an entry from the node directory" on page 448	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
Node directory management APIs	"sqlenops - Start a node directory scan" on page 432	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
Node directory management APIs	"sqlengne - Get the next node directory entry" on page 430	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
Node directory management APIs	"sqlencls - End a node directory scan" on page 429	sqlenv	C: ininfo.c C++: ininfo.C COBOL: nodecat.cbl
Node directory management APIs	"db2UpdateAlternateServerForDB - Update the alternate server for a database alias in the system database directory" on page 324	db2ApiDf	n/a
Satellite synchronization APIs	"db2GetSyncSession - Get a satellite synchronization session identifier" on page 123	db2ApiDf	n/a
Satellite synchronization APIs	"db2QuerySatelliteProgress - Get the status of a satellite synchronization session" on page 237	db2ApiDf	n/a

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

API Type	DB2 API	Include File	Sample Programs
<b>Satellite synchronization APIs</b>	"db2SetSyncSession - Set satellite synchronization session" on page 309	db2ApiDf	n/a
<b>Satellite synchronization APIs</b>	"db2SyncSatellite - Start satellite synchronization" on page 315	db2ApiDf	n/a
<b>Satellite synchronization APIs</b>	"db2SyncSatelliteStop - Pause satellite synchronization" on page 316	db2ApiDf	n/a
<b>Satellite synchronization APIs</b>	"db2SyncSatelliteTest - Test whether a satellite can be synchronized" on page 317	db2ApiDf	n/a
<b>Read log files APIs</b>	"db2ReadLog - Read log records" on page 239	db2ApiDf	C: dbrecov.sqc C++: dbrecov.sqC
<b>Read log files APIs</b>	"db2ReadLogNoConn - Read the database logs without a database connection" on page 245	db2ApiDf	n/a
<b>Read log files APIs</b>	"db2ReadLogNoConnInit - Initialize reading the database logs without a database connection" on page 248	db2ApiDf	n/a
<b>Read log files APIs</b>	"db2ReadLogNoConnTerm - Terminate reading the database logs without a database connection" on page 250	db2ApiDf	n/a
<b>Indoubt transaction management APIs</b>	"db2XaListIndTrans - List indoubt transactions" on page 474	db2ApiDf	n/a
<b>Indoubt transaction management APIs</b>	"sqlxhfrg - Forget transaction status" on page 479	sqlxa	n/a
<b>Indoubt transaction management APIs</b>	"sqlxphcm - Commit an indoubt transaction" on page 480	sqlxa	n/a
<b>Indoubt transaction management APIs</b>	"sqlxphrl - Roll back an indoubt transaction" on page 481	sqlxa	n/a
<b>Indoubt transaction management APIs</b>	"sqlcspqy - List DRDA indoubt transactions" on page 366	sqlxa	n/a
<b>APIs for obtaining concurrent access to a database</b>	"sqleAttachToCtx - Attach to context" on page 484	sql	C: dbthrds.sqc C++: dbthrds.sqC
<b>APIs for obtaining concurrent access to a database</b>	"sqleBeginCtx - Create and attach to an application context" on page 485	sql	C: dbthrds.sqc C++: dbthrds.sqC

## DB2 APIs

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	"sqlDetachFromCtx - Detach from context" on page 486	sql	C: dbthrds.sqc C++: dbthrds.sqC
APIs for obtaining concurrent access to a database	"sqlEndCtx - Detach from and free the memory associated with an application context" on page 487	sql	n/a
APIs for obtaining concurrent access to a database	"sqlGetCurrentCtx - Get current context" on page 489	sql	n/a
APIs for obtaining concurrent access to a database	"sqlInterruptCtx - Interrupt context" on page 490	sql	n/a
APIs for obtaining concurrent access to a database	"sqlSetTypeCtx - Set application context type" on page 491	sql	C: dbthrds.sqc C++: dbthrds.sqC
Database partition management APIs	"sqladdn - Add a database partition to the partitioned database environment" on page 373	sqlenv	n/a
Database partition management APIs	"sqldrpn - Check whether a database partition server can be dropped" on page 406	sqlenv	n/a
Database partition management APIs	"sqlcran - Create a database on a database partition server" on page 387	sqlenv	n/a
Database partition management APIs	"sqldpan - Drop a database on a database partition server" on page 402	sqlenv	n/a
Database partition management APIs	"sqlsdeg - Set the maximum runtime intrapartition parallelism level or degree for SQL statements" on page 439	sqlenv	C: ininfo.c C++: ininfo.C
Database partition management APIs	"sqlgtpi - Get table distribution information" on page 462	sqlutil	n/a
Database partition management APIs	"sqlgrpn - Get the database partition server number for a row" on page 459	sqlutil	n/a
Miscellaneous APIs	"db2AdminMsgWrite - Write log messages for administration and replication function" on page 40	db2ApiDf	n/a



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

API Type	DB2 API	Include File	Sample Programs
<b>Miscellaneous APIs</b>	“db2XaGetInfo - Get information for a resource manager” on page 473	sqlxa	n/a
<p><b>Note:</b> 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><b>C/C++ (UNIX):</b>            sqllib/include</p> <p><b>C/C++ (Windows):</b>            sqllib\include</p> <p><b>COBOL (UNIX):</b>            sqllib/include/cobol_a            sqllib/include/cobol_i            sqllib/include/cobol_mf</p> <p><b>COBOL (Windows):</b>            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"> <li>• sqlesetc API - Set client connection settings</li> <li>• sqleseti API - Set client information</li> <li>• sqleqryc API - Query client connection settings</li> <li>• sqleqryi API - Query client information</li> </ul>
dbcfg.sqc, dbcfg.sqC	<ul style="list-style-type: none"> <li>• db2AutoConfig API - Access the Configuration Advisor</li> <li>• db2AutoConfigFreeMemory API - Free the memory allocated by the db2AutoConfig API</li> <li>• sqlesetc API - Set client connection settings</li> <li>• sqlaintp API - Get error message</li> </ul>
dbconn.sqc, dbconn.sqC	<ul style="list-style-type: none"> <li>• db2DatabaseRestart API - Restart database</li> <li>• sqlefrce API - Force users and applications off the system</li> </ul>
dbcreate.c, dbcreate.C	<ul style="list-style-type: none"> <li>• sqlecrea API - Create database</li> <li>• sqledrpd API - Drop database</li> </ul>
dbinfo.c, dbinfo.C	<ul style="list-style-type: none"> <li>• db2CfgGet API - Get the database manager or database configuration parameters</li> <li>• db2CfgSet API - Set the database manager or database configuration parameters</li> </ul>
dbmcon.sqc, dbmcon.sqC	<ul style="list-style-type: none"> <li>• sqlesetc API - Set client connection settings</li> </ul>
dbmigrat.c, dbmigrat.C	<ul style="list-style-type: none"> <li>• sqlemgdb API - Migrate previous version of DB2 database to current version</li> </ul>
dbpkg.sqc, dbpkg.sqC	<ul style="list-style-type: none"> <li>• sqlaprep API - Precompile application program</li> <li>• sqlabndx API - Bind application program to create a package</li> <li>• sqlarbnd API - Rebind package</li> </ul>

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

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

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

Sample Program	Included APIs
ininfo.c, ininfo.C	<ul style="list-style-type: none"> <li>• db2CfgGet API - Get the database manager or database configuration parameters</li> <li>• db2CfgSet API - Set the database manager or database configuration parameters</li> <li>• sqlgins API - Get current instance</li> <li>• sqlctnd API - Catalog an entry in the node directory</li> <li>• sqlenops API - Start a node directory scan</li> <li>• sqlengne API - Get the next node directory entry</li> <li>• sqlencls API - End a node directory scan</li> <li>• sqlleuncl API - Uncatalog an entry from the node directory</li> <li>• sqlcadb API - Catalog a database in the system database directory</li> <li>• db2DbDirOpenScan API - Start a system or local database directory scan</li> <li>• db2DbDirGetNextEntry API - Get the next system or local database directory entry</li> <li>• sqledcgs API - Change a database comment in the system or local database directory</li> <li>• db2DbDirCloseScan API - End a system or local database directory scan</li> <li>• sqlleuncl API - Uncatalog a database from the system database directory</li> <li>• sqlcgdad API - Catalog a database in the database connection services (DCS) directory</li> <li>• sqlcgdsc API - Start a database connection services (DCS) directory scan</li> <li>• sqlcgdge API - Get a specific entry in the database connection services (DCS) directory</li> <li>• sqlcgdgt API - Get database connection services (DCS) directory entries</li> <li>• sqlcgdcl API - End a database connection services (DCS) directory scan</li> <li>• sqlcgdel API - Uncatalog a database from the database connection services (DCS) directory</li> <li>• sqlcsdeg API - Set the maximum runtime intrapartition parallelism level or degree for SQL statements</li> </ul>
instart.c, instart.C	<ul style="list-style-type: none"> <li>• sqlfrce API - Force users and applications off the system</li> <li>• db2InstanceStart API - Start instance</li> <li>• db2InstanceStop API - Stop instance</li> </ul>
tbmove.sqc, tbmove.sqC	<ul style="list-style-type: none"> <li>• db2Export API - Export data from a database</li> <li>• db2Import API - Import data into a table, hierarchy, nickname or view</li> <li>• sqluvqdp API - Quiesce table spaces for a table</li> <li>• db2Load API - Load data into a table</li> <li>• db2LoadQuery API - Get the status of a load operation</li> </ul>
tbreorg.sqc, tbreorg.sqC	<ul style="list-style-type: none"> <li>• db2Reorg API - Reorganize an index or a table</li> <li>• db2Runstats API - Update statistics about the characteristics of a table and associated indexes</li> </ul>

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

Sample Program	Included APIs
tscreate.sqc, tscreate.sqC	<ul style="list-style-type: none"> <li>db2CfgGet API - Get the database manager or database configuration parameters</li> </ul>
utilapi.c, utilapi.C	<ul style="list-style-type: none"> <li>sqlaintp API - Get error message</li> <li>sqllogstt API - Get the SQLSTATE message</li> <li>sqleatin API - Attach to instance</li> <li>sqledtin API - Detach from instance</li> </ul>
utilsnap.c, utilsnap.C	<ul style="list-style-type: none"> <li>db2GetSnapshot API - Get a snapshot of the database manager operational status</li> <li>db2MonitorSwitches API - Get or update the monitor switch settings</li> </ul>

Table 3. COBOL Sample Programs with DB2 APIs

Sample Program	Included APIs
checkerr.cbl	<ul style="list-style-type: none"> <li>sqlaintp API - Get error message</li> <li>sqllogstt API - Get the SQLSTATE message</li> </ul>
client.cbl	<ul style="list-style-type: none"> <li>sqleqryc API - Query client connection settings</li> <li>sqlesetc API - Set client connection settings</li> </ul>
db_udcs.cbl	<ul style="list-style-type: none"> <li>sqleatin API - Attach to instance</li> <li>sqlecrea API - Create database</li> <li>sqledrpd API - Drop database</li> </ul>
dbcatt.cbl	<ul style="list-style-type: none"> <li>sqlecadb API - Catalog a database in the system database directory</li> <li>db2DbDirCloseScan API - End a system or local database directory scan</li> <li>db2DbDirGetNextEntry API - Get the next system or local database directory entry</li> <li>db2DbDirOpenScan API - Start a system or local database directory scan</li> <li>sqleuncd API - Uncatalog a database from the system database directory</li> </ul>
dbcmt.cbl	<ul style="list-style-type: none"> <li>sqledcgd API - Change a database comment in the system or local database directory</li> <li>db2DbDirCloseScan API - End a system or local database directory scan</li> <li>db2DbDirGetNextEntry API - Get the next system or local database directory entry</li> <li>db2DbDirOpenScan API - Start a system or local database directory scan</li> <li>sqleisig API - Install signal handler</li> </ul>
dbinst.cbl	<ul style="list-style-type: none"> <li>sqleatcp API - Attach to instance and change password</li> <li>sqleatin API - Attach to instance</li> <li>sqledtin API - Detach from instance</li> <li>sqlegins API - Get current instance</li> </ul>

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

Sample Program	Included APIs
dbstat.sqb	<ul style="list-style-type: none"> <li>db2Reorg API - Reorganize an index or a table</li> <li>db2Runstats API - Update statistics about the characteristics of a table and associated indexes</li> </ul>
dcscat.cbl	<ul style="list-style-type: none"> <li>sqlgdad API - Catalog a database in the database connection services (DCS) directory</li> <li>sqlgdcl API - End a database connection services (DCS) directory scan</li> <li>sqlgdel API - Uncatalog a database from the database connection services (DCS) directory</li> <li>sqlgdge API - Get a specific entry in the database connection services (DCS) directory</li> <li>sqlgdgt API - Get database connection services (DCS) directory entries</li> <li>sqlgdsc API - Start a database connection services (DCS) directory scan</li> </ul>
ebcdicdb.cbl	<ul style="list-style-type: none"> <li>sqleatin API - Attach to instance</li> <li>sqlecrea API - Create database</li> <li>sqledrpd API - Drop database</li> </ul>
exp samp.sqb	<ul style="list-style-type: none"> <li>db2Export API - Export data from a database</li> <li>db2Import API - Import data into a table, hierarchy, nickname or view</li> </ul>
impexp.sqb	<ul style="list-style-type: none"> <li>db2Export API - Export data from a database</li> <li>db2Import API - Import data into a table, hierarchy, nickname or view</li> </ul>
loadqry.sqb	<ul style="list-style-type: none"> <li>db2LoadQuery API - Get the status of a load operation</li> </ul>
migrate.cbl	<ul style="list-style-type: none"> <li>sqlmgdb API - Migrate previous version of DB2 database to current version</li> </ul>
nodecat.cbl	<ul style="list-style-type: none"> <li>sqlctnd API - Catalog an entry in the node directory</li> <li>sqlencls API - End a node directory scan</li> <li>sqlengne API - Get the next node directory entry</li> <li>sqlenops API - Start a node directory scan</li> <li>sqluncn API - Uncatalog an entry from the node directory</li> </ul>
rebind.sqb	<ul style="list-style-type: none"> <li>sqlarbd API - Rebind package</li> </ul>
tabscont.sqb	<ul style="list-style-type: none"> <li>sqlbctq API - Close a table space container query</li> <li>sqlbftq API - Fetch the query data for rows in a table space container</li> <li>sqlbotq API - Open a table space container query</li> <li>sqlbtcq API - Get the query data for all table space containers</li> <li>sqlfmem API - Free the memory allocated by the sqlbctq and sqlbmtsq APIs</li> </ul>

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

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

---

## Chapter 2. Changed APIs and data structures

This section provides information about the changed APIs and data structures.

**Note:** DB2 offers binary compatibility for existing application, as well as compatibility with earlier versions of scripts, SQL queries, and other supported interfaces. However, there might be compatibility issues with source code files, as API structures and values might change.

*Table 4. Back-level supported APIs and data structures*

API or Data Structure (Version)	Descriptive Name	New API, Data Structure, or Table Function (Version)
sqlbftsqs (V2)	Fetch Table Space Query	sqlbftpq (V5)
sqlbstsq (V2)	Single Table Space Query	sqlbstpq (V5)
sqlbtsq (V2)	Table Space Query	sqlbmstsq (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)
sqlpststart (V5)	Start Database Manager	db2InstanceStart (V8)
sqlpststop (V5)	Stop Database Manager	db2InstanceStop (V8)
sqlpststr (V2)	Start Database Manager (DB2 Parallel Edition Version 1.2)	db2InstanceStart (V8)
sqlpststar (V2)	Start Database Manager (DB2 Version 2)	db2InstanceStart (V8)
sqlpststop (V2)	Stop Database Manager	db2InstanceStop (V8)
sqlrststd (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)
sqlfrs (V7)	Reset Database Manager Configuration	db2CfgSet (V8)
sqlfudb (V7)	Update Database Configuration	db2CfgSet (V8)
sqlfus (V7)	Update Database Manager Configuration	db2CfgSet (V8)
sqlfxdb (V7)	Get Database Configuration	db2CfgGet (V8)
sqlfxs (V7)	Get Database Configuration	db2CfgGet (V8)
sqlmon (V6)	Get/Update Monitor Switches	db2MonitorSwitches (V7)
sqlmonss (V5)	Get Snapshot	db2GetSnapshot (V6)
sqlmonsz (V6)	Estimate Size Required for sqlmonss() Output Buffer	db2GetSnapshotSize (V7)
sqlmrset (V6)	Reset Monitor	db2ResetMonitor (V7)
sqlubkp (V5)	Backup Database	db2Backup (V8)



## Changed APIs and data structures

Table 4. Back-level supported APIs and data structures (continued)

API or Data Structure (Version)	Descriptive Name	New API, Data Structure, or Table Function (Version)
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)

Table 4. Back-level supported APIs and data structures (continued)

API or Data Structure (Version)	Descriptive Name	New API, Data Structure, or Table Function (Version)
db2DbDirNextEntryStruct (V8.2)	Get Next Database Directory Entry data structure	db2DbDirNextEntryStructV9 (V9.1)
db2gDbDirNextEntryStruct (V8.2)	Get Next Database Directory Entry data structure	db2gDbDirNextEntryStrV9 (V9.1)
sqlbctsq	Close a table space query	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)
sqlbotsq	Open a table space query	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)
sqlbftpq	Fetch the query data for rows in a table space	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)
sqlbgtss	Get table space usage statistics	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)
sqlbmtsq	Get the query data for all table spaces	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)
sqlbstpq	Get information about a single table space	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)
sqlbtcq	Get the query data for all table space containers	MON_GET_TABLESPACE and the MON_GET_CONTAINER table functions (V9.7)

Table 5. Back-level unsupported APIs and data structures

Name	Descriptive Name	Replacement API, data structure or table function
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
sqluadaw	Get current user's authorities	AUTH_LIST_AUTHORITIES_FOR_AUTHID table function
SQL-AUTHORIZATIONS	Authorizations structure	none required

## Changed APIs and data structures

---

## Chapter 3. Log sequence number changes affecting API and application behavior

The size of the log sequence number (LSN) has increased from six-bytes to eight-bytes. A new data type, db2LSN, has been created in support of the new LSN size.

A number of APIs with input and output structures containing LSN fields have been updated to use the new db2LSN data type. The updated APIs are discussed in the following sections. To ensure that the latest versions of the APIs are used, the new DB2 version number constant must be passed to the APIs through the version number API input parameter.

Resulting from the LSN size increase and the creation of new versions of the APIs, the behavior of the affected APIs is dependent on the current client-server configuration and the level of the applications in use. Under certain scenarios there are restrictions and conditions to take note of regarding the extent of the APIs abilities.

**Note:** For the older APIs listed in the following sections, these APIs do not have new versions and therefore their structures do not make use of the db2LSN data type, though their behavior is still influenced by the client-server configuration being used.

### db2ReadLog API

The behavior of the db2ReadLog API changes depending on the client-server configuration.

Here are the possible configurations:

- Latest client and latest server: An error message SQL2032N is returned if existing applications make use of the older version of the db2ReadLog API. Calls to the older version of the db2ReadLog API are *not supported*. Such applications must be updated to make use of the latest version of the db2ReadLog API. If you are using an application developed with the latest version of the db2ReadLog API, then there are no restrictions on the functions of the API.

**Note:** If the client and server are on differing endian platforms, then all native data type fields in the db2ReadLogInfoStruct output structure are byte-reversed, *including* the LSN fields.

- Older client and latest server: As with the previous configuration, older level db2ReadLog API calls are not supported and the error message SQL2032N is returned. The client must be upgraded and any existing applications updated to use the latest version of the db2ReadLog API in order to proceed.
- Latest client and older server: Whether the latest version of the db2ReadLog API is invoked from within a newly developed application, or an older version db2ReadLog API call is made from an existing application, there are no restrictions on the functions of the API. In both cases however, the log records returned in the log buffer are representative of the old log records, with the old LSN format.

## Log sequence number changes affecting API and application behavior

**Note:** If the client and server are on differing endian platforms, then all native data type fields in the db2ReadLogInfoStruct output structure are byte-reversed, *excluding* the LSN fields.

### db2ReadLogNoConn API

The behavior of the db2ReadLogNoConn API changes depending on the client-server configuration.

Here are the possible configurations:

- Latest client and latest server: An error message SQL2032N is returned if existing applications make use of the older version of the db2ReadLogNoConn API. Calls to the older version of the db2ReadLogNoConn API are *not supported*. Such applications must be updated to make use of the latest version of the db2ReadLogNoConn API. If you are using an application developed with the latest version of the db2ReadLogNoConn API, then there are no restrictions on the functions of the API.

### db2HistoryGetEntry API

The behavior of the db2HistoryGetEntry API changes depending on the client-server configuration.

Here are the possible configurations:

- Latest client and latest server: An error message SQL2032N is returned if existing applications make use of the older version of the db2HistoryGetEntry API and if the LSNs of the history file entries are larger than can be contained by the “oLastLSN” field of the older level API output structure. Such applications must be updated to make use of the latest version of the db2HistoryGetEntry API to correct this behavior. If you are using an application developed with the latest version of the db2HistoryGetEntry API, then there are no restrictions on the functions of the API.
- Older client and latest server: An error message SQL2032N is returned if an older version of the db2HistoryGetEntry API is used, and if the LSNs of the history file entries are larger than can be contained by the **oLastLSN** field of the older level API output structure. An upgrade of the client and an update of the existing applications updated are needed to correct the problem.
- Latest client and older server: If the latest version of the db2HistoryGetEntry API is invoked from within a newly developed application, or if a call to the earlier version of the db2HistoryGetEntry API is made from an existing application, then there are no restrictions on the functions of the API.

### db2Prune API

The behavior of the db2Prune API changes depending on the client-server configuration.

Here are the possible configurations:

- Latest client and latest server: If you are using a new application with the latest version of the db2Prune API, or you are using an existing application that uses an earlier version of the db2Prune API, then there are no restrictions on the functions of the API.

## Log sequence number changes affecting API and application behavior

- Older client and latest server: If you are using an existing application and making calls to an earlier version of the db2Prune API, then there are no restrictions on the functions of the API.
- Latest client and older server: When using the latest version of the db2Prune API with an LSN string as input, an error message SQL2032N is returned if the LSN string either represents an LSN that exceeds the value 0xFFFF FFFF FFFF, or the LSN string is less than 12 characters in length. To bypass this error, the server must be upgraded to at least match the level of the client. If making calls to an older version of the db2Prune API through an existing application, there are no restrictions on the functions of the API as introduced by the changes to the LSN.

### sqlbftpq API, sqlbstpq API, and sqlbmtsq API

The behavior of the sqlbftpq API, sqlbstpq API, and sqlbmtsq API changes depending on the client-server configuration.

**Note:** These APIs have been deprecated and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information.

Here are the possible configurations:

- Latest client and latest server: For each of these APIs, if calls are made to an earlier version of the API through an existing application, an error message SQL2032N is returned, provided that the LSN returned is larger in size than the **lifeLSN** field of the older SQLB\_TBSPQRY\_DATA structure can contain. There is an additional consideration for the sqlbmtsq API: when making a call to an earlier version of this API, the **lifeLSN** field of the SQLB\_TBSPQRY\_DATA structure will contain an invalid value, even though all other fields of the structure contain valid values. Updating the applications to use the latest versions of the APIs can solve these problems. If using the latest versions of these APIs, then there are no restrictions on the functions of the APIs.
- Older client and latest server: For this configuration, exercising the earlier version of any of the three APIs results in a behavior that mirrors the behavior of the APIs in the configuration where both the client and server are of the latest release, as noted previously, excluding the limitation noted for the sqlbmtsq API. To resolve these limitations, the client must be upgraded to at least the release level of the server, and the applications in use updated to make use of the latest versions of the APIs.
- Latest client and older server: When making a call to an earlier version of the sqlbmtsq API through an existing application, the **lifeLSN** field of the SQLB\_TBSPQRY\_DATA structure is returned with an invalid LSN value. All other fields of the structure remain valid. To correct this behavior, the application in use must be updated to incorporate the latest version of the sqlbmtsq API. For the other APIs, there are no restrictions on the functions of the APIs regardless of the version of the API in use.

### sqlurlog API (older level API)

The behavior of the sqlurlog API changes depending on the client-server configuration.

Here are the possible configurations:

- Latest client and latest server: Using the sqlurlog API in this client-server configuration could result in an error message SQL2650N being returned should

## Log sequence number changes affecting API and application behavior

any of the LSNs returned be greater in value than can be contained by the LSN fields of the SQLU\_RLOG\_INFO output structure. The only way to avoid this problem would be to modify any applications using the sqlurlog API to make use of its successor, the db2ReadLog API.

**Note:** Using the sqlurlog API in this configuration, the log records returned through the log buffer are representative of the new log records and contain LSNs of the new db2LSN format.

- Older client and latest server: Similar to the db2ReadLog API and the db2ReadLogNoConn API, sqlurlog API calls issued from the older level clients against current level servers are “not supported”. Any attempts at using the sqlurlog API result in the return of the error message SQL1198N. To avoid this problem, the client must be upgraded to match the level of the server.
- Latest client and older server: There are no restrictions on the functions of the API when using the sqlurlog API in this client-server configuration.

### sqlbftsq API, sqlbstsq API, and sqlbtsq API (older level APIs)

The behavior of the sqlbftsq API, sqlbstsq API, and sqlbtsq API changes depending on the client-server configuration.

Here are the possible configurations:

- Latest client and latest server: Using any of these three APIs in this type of client-server configuration could result in the population of the **lifeLSN** field of the older SQLB\_TBSPQRY\_DATA structure with an invalid LSN value. This is a consequence of the **lifeLSN** field of the SQLB\_TBSPQRY\_DATA structure being unable to contain large LSN values. To avoid this problem, the successors of the sqlbftsq API, sqlbstsq API, and sqlbtsq API must be used, where the successors are the sqlbftpq API, sqlbstpq API, and sqlbmstsq API.
- Older client and latest server: The API behavior described in the previous configuration applies for this configuration as well.
- Latest client and older server: There are no limitations in the functions of the APIs when using the sqlbftsq API, sqlbstsq API, and sqlbtsq API under this client-server configuration.



---

## Chapter 4. How the API descriptions are organized

This section provides information about 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 requirements: 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`

## How the API descriptions are organized

### COBOL (Windows):

```
sqllib\include\cobol_a  
sqllib\include\cobol_i  
sqllib\include\cobol_mf
```

## C API syntax

The C syntax of the API call.

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

- The new API names contain the prefix "db2", followed by a meaningful mixed case string (for example, db2LoadQuery). Related APIs have names that allow them to be logically grouped. For example:

```
db2HistoryCloseScan  
db2HistoryGetEntry  
db2HistoryOpenScan  
db2HistoryUpdate
```

- Generic APIs have names that contain the prefix "db2g", followed by a string that matches the C API name. Data structures used by generic APIs have names that also contain the prefix "db2g".
- The first parameter into the function (*versionNumber*) represents the version, release, or PTF level to which the code is to be compiled. This version number is used to specify the level of the structure that is passed in as the second parameter.
- The second parameter into the function is a void pointer to the primary interface structure for the API. Each element in the structure is either an atomic type (for example, db2Long32) or a pointer. Each parameter name adheres to the following naming conventions:

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

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

## Generic API syntax

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

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

## API parameters

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

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

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

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

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

### REXX API syntax

The REXX syntax of the API call, where appropriate.

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

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

### REXX API parameters

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

### 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 in the following list:

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.

FORTTRAN 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.

**Include files for DB2 API applications**

---

## Chapter 5. Administrative APIs

---

## 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.



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

### **iType**

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

- DB2CONTACT\_EMAIL
- DB2CONTACT\_PAGE

### **piAddress**

Input. The email 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.

---

### 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.

---

## db2AddSnapshotRequest - Add a snapshot request

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 second parameter **pParmStruct**. To use the structure db2AddSnapshotData as described previously, specify db2Versio910. If you want to use a different

## db2AddSnapshotRequest - Add a snapshot request

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 or output, or both. 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.

### db2AdminMsgWrite - Write log messages for administration and replication function

Provides a mechanism for users and Replication to write information to the **db2diag** log file, 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 the **db2diag** log file 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 file, unless the **diaglevel** database manager configuration parameter is set to zero.

---

## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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****iaAliasLen**

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

**iuUserNameLen**

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

**ipPasswordLen**

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

---

# db2AutoConfig - Access the Configuration Advisor

Allows application programs to access the Configuration Advisor.

## 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;

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:

##### DB2\_SG\_NOT\_APPLY

Do not apply any recommendations

## db2AutoConfig - Access the Configuration Advisor

### 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.

### 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 before 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 before 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 before 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.

With the deprecation of the **maxagents** and **maxcagents** configuration parameters, the behavior of the db2AutoConfig API will depend on the **db2VersionNumber** passed in to the API. If the version is DB2 v9.5 or beyond, **maxagents** will not be returned, but, for versions before this, it will. In a future release, these configuration parameters may be removed completely.

## 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.

## db2Backup - Back up a database or table space

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

### Scope

In a partitioned database environment, by default this API affects only the database partition on which it is executed.

If the option to perform a partitioned backup is specified, the command can be called only on the catalog node. If the option specifies that all database partition servers are to be backed up, it affects all database partition servers that are listed in the `db2nodes.cfg` file. Otherwise, it affects the database partition servers that are specified on the API.

### Authorization

One of the following authorities:

- *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;
```

## db2Backup - Back up a database or table space

```
    db2int32 iAllNodeFlag;
    db2int32 iNumNodes;
    db2NodeType *piNodeList;
    db2int32 iNumMPPOutputStructs;
    struct db2BackupMPPOutputStruct *poMPPOutputStruct;
} db2BackupStruct;

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

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

typedef SQL_STRUCTURE db2BackupMPPOutputStruct
{
    db2NodeType nodeNumber;
    db2UInt64 backupSize;
    struct sqlca sqlca;
} db2BackupMPPOutputStruct;

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;
    db2int32 iAllNodeFlag;
    db2int32 iNumNodes;
    db2NodeType *piNodeList;
    db2int32 iNumMPPOutputStructs;
    struct db2gBackupMPPOutputStruct *poMPPOutputStruct;
} db2gBackupStruct;
```



## db2Backup - Back up a database or table space

```
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 db2gBackupMPPOutputStruct
{
    db2NodeType nodeNumber;
    db2UInt64 backupSize;
    struct sqlca sqlca;
} db2gBackupMPPOutputStruct;

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**.

#### 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. 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.

## db2Backup - Back up a database or table space

### **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 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 required.

#### **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\_DEDUP\_DEVICE**

Optimizes the format of the backup image for target storage devices that support data deduplication.

#### **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

## db2Backup - Back up a database or table space

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, with the exception of snapshot backups where INCLUDE is the default.

### DB2BACKUP\_MPP

Perform backup in a manner suitable for a partitioned database.

### iUtilImpactPriority

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

- 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 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.

### iAllNodeFlag

Input. Partitioned database environments only. Indicates whether the backup operation is to be applied to all or some 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.

### **DB2\_ALL\_EXCEPT**

Apply to all database partition servers except those in a list that is passed in **piNodeList**.

### **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 backup.

### **iNumMPPOutputStructs**

Input. Specifies the number of elements in the **piMPPOutputStruct** array. This must be equal to or greater than the number of database partition servers that participate in this backup operation.

### **piMPPOutputStruct**

Output. A pointer to an array of db2BackupMPPOutputStruct structures that specify output parameters for particular database partition servers.

## **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: 'L'**

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

#### **SQLU\_XBSA\_MEDIA: 'X'**

XBSA interface.

#### **SQLU\_TSM\_MEDIA: 'A'**

Tivoli® Storage Manager.

#### **SQLU\_OTHER\_MEDIA: 'O'**

Vendor library.

#### **SQLU\_SNAPSHOT\_MEDIA: 'F'**

Specifies that a plug-in library snapshot backup is to be taken.

You cannot use SQLU\_SNAPSHOT\_MEDIA with any of the following options:

- DB2BACKUP\_COMPRESS

## db2Backup - Back up a database or table space

- **DB2BACKUP\_TABLESPACE**
- **DB2BACKUP\_INCREMENTAL**
- **iNumBuffers**
- **iBufferSize**
- **iParallelism**
- **piComprOptions**
- **iUtilImpactPriority**
- **numLocations** field of this structure must be 1 for snapshot restore

The default behavior for a snapshot backup is a FULL DATABASE OFFLINE backup of all paths that make up the database including all containers, local volume directory, database path (DBPATH), and primary log and mirror log paths (**INCLUDE LOGS** is the default for all snapshot backups unless **EXCLUDE LOGS** is explicitly stated).

- IBM® TotalStorage SAN Volume Controller
- IBM Enterprise Storage Server® Model 800
- IBM Storwize® v7000
- IBM System Storage® DS6000™
- IBM System Storage DS8000®
- IBM System Storage N Series
- IBM XIV®

**SQLU\_SNAPSHOT\_SCRIPT\_MEDIA: 'f'**

Specifies that a scripted snapshot backup is to be taken.

## db2BackupMPPOutputStruct and db2gBackupMPPOutputStruct data structure parameters

### **nodeNumber**

The database partition server to which the option applies.

### **backupSize**

The size of the backup on the specified database partition, in kilobytes.

**sqlca** The sqlca from the specified database partition.

## 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**

You can only perform a snapshot backup with **versionNumber** db2Version950 or higher. If you specify media type `SQLU_SNAPSHOT_MEDIA` with a **versionNumber** lower than db2Version950, DB2 database will return an error.

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.

As you regularly backup your database, you might accumulate very large database backup images, many database logs and load copy images, all of which might be taking up a large amount of disk space. Refer to “Managing recovery objects” for information about how to manage these recovery objects.

### **Usage notes for a single system view (SSV) backup in a partitioned database environment**

- To perform an SSV backup, specify **iOptions** `DB2BACKUP_MPP` and one of `DB2BACKUP_DB` or `DB2BACKUP_TABLESPACE`.
- You can only perform a SSV backup with **versionNumber** db2Version950 or higher. If you specify **iOptions** `DB2BACKUP_MPP` with a **versionNumber** lower than db2Version950, DB2 database will return an error. If you specify other options related to SSV backup with a **versionNumber** lower than db2Version950, DB2 database will ignore those options.
- The values for **piMediaList** specified directly in `db2BackupStruct` will be used as the default values on all nodes.
- The value of **oBackupSize** returned in the `db2BackupStruct` is the sum of the backup sizes on all nodes. The value of **backupSize** returned in the `db2BackupMPPOutputStruct` is the size of the backup on the specified database partition.
- **iAllNodeFlag**, **iNumNodes**, and **piNodeList** operate the same as the similarly-named elements in `db2RollforwardStruct`, with the exception that there is no `CAT_NODE_ONLY` value for **iAllNodeFlag**.
- SSV backups performed with the `DB2BACKUP_BACKUP` caller action are performed as if the `DB2BACKUP_NOINTERRUPT` caller action was specified.

## db2Backup - Back up a database or table space

- **\*poMPPOutputStruct** points to memory allocated by the caller that contains at least as many elements as there are database partitions participating in the backup.



## 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;
    char *dbname;
```

## db2CfgGet - Get the database manager or database configuration parameters

```
} 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.

#### 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".

## db2CfgGet - Get the database manager or database configuration parameters

**Note:** Additional db2Cfg tokens are added to support getting (or setting) of cluster caching facility configuration parameters and cluster caching facility structure configuration parameters in a DB2 pureScale® environment.

### **ptrvalue**

Output. The configuration parameter value.

### **flags**

Output. 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.

#### **db2CfgParamComputed**

Indicates whether the retrieved parameter has a value of COMPUTED. To determine whether a given configuration parameter has been set to COMPUTED, perform a boolean AND operation against the value returned by the flag and the **db2CfgParamComputed** keyword defined in db2ApiDf.h.

If the boolean AND operation is false for both of these keywords, it means that the retrieved parameter value is set manually.

## **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.

## **Usage notes**

The configuration parameters **maxagents** and **maxcagents** are deprecated. In a future release, these configuration parameters may be removed completely.

The db2CfgGet API will tolerate requests for SQLF\_KTN\_MAXAGENTS and SQLF\_KTN\_MAXCAGENTS, but 0 will be returned if the server is DB2 V9.5.

### 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 database partition on which the database has been created.

#### Scope

Modifications to the database configuration file affect all database partitions by default.

#### Authorization

For modifications to the database configuration file, one of the following authorities:

- 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;
    SQL_PDB_NODE_TYPE dbpartitionnum;
    SQL_PDB_NODE_TYPE member;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
    db2UInt32 token;
    char *ptrvalue;
    db2UInt32 flags;
} db2CfgParam;
```

## db2CfgSet - Set the database manager or database configuration parameters

```
SQL_API_RC SQL_API_FN
db2gCfgSet (
    db2UInt32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
    db2UInt32 numItems;
    struct db2gCfgParam *paramArray;
    db2UInt32 flags;
    db2UInt32 dbname_len;
    char *dbname;
} db2gCfg;

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

### 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.

## db2CfgSet - Set the database manager or database configuration parameters

### **db2CfgReset**

Reset all database manager configuration parameters to default values.

### **db2CfgSingleDbpartition**

To update or reset the database configuration on a specific database partition, set this flag and provide a value for **dbpartitionnum**. This is only applicable in a partitioned database environment and ignored in other environments.

### **db2CfgSingleMember**

To update or reset the database configuration on a specific member, set this flag and provide a value for member. This is only applicable in a DB2 pureScale environment and ignored in other environments.

### **dbname**

Input. The database name.

### **dbpartitionnum**

Input. Specifies on which database partition this API will set the configuration value. The **DBPARTITIONNUM** parameter is only applicable in a partitioned database environment and must be 0 in a DB2 pureScale environment.

### **member**

Input. Specifies on which member this API will set the configuration value. The **MEMBER** parameter is only applicable in a DB2 pureScale environment and ignored in other environments.

## **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".

**Note:** Additional db2Cfg tokens are added to support getting (or setting) of cluster caching facility configuration parameters and cluster caching facility structure configuration parameters in a DB2 pureScale environment.

### **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.

#### **db2CfgParamComputed**

Indicates whether the retrieved parameter has a value of computed. To determine whether a given configuration parameter has been set to computed, perform a boolean AND operation against the value returned by the flag and the **db2CfgParamComputed** keyword defined in db2ApiDf.h.

#### **db2CfgParamManual**

Used to unset the automatic or computed setting of a parameter and set the parameter to the current value. The **ptrvalue** field is ignored and can be set to NULL.

## db2CfgSet - Set the database manager or database configuration parameters

### 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.

### Usage notes

The configuration parameters **maxagents** and **maxcagents** are deprecated. In a future release, these configuration parameters may be removed completely.

The db2CfgSet API will tolerate updates to the **maxagents** and **maxcagents** configuration parameters, however these updates will be ignored by DB2.

### Usage samples

CASE 1: The **MAXAPPLS** parameter will be set to 50 at dbpartitionnum 30.

CASE 2: The **MAXAPPLS** parameter will be set to 50 on all dbpartitionnum.

```
int updateDbConfig()
{
    struct sqlca sqlca = {0};
    db2Cfg cfgStruct = {0};
    db2CfgParam cfgParameters[2];
    char *dbAlias = "SAMPLE";

    /* initialize cfgParameters */
    cfgParameters[0].flags = 0;
    cfgParameters[0].token = SQLF_DBTN_TSM_OWNER;
    cfgParameters[0].ptrvalue = (char *)malloc(sizeof(char) * 65);
    cfgParameters[1].flags = 0;
    cfgParameters[1].token = SQLF_DBTN_MAXAPPLS;
    cfgParameters[1].ptrvalue = (char *)malloc(sizeof(sqluint16));

    /* set two DB Config. fields */
    strcpy(cfgParameters[0].ptrvalue, "tsm_owner");
    *(sqluint16 *) (cfgParameters[1].ptrvalue) = 50;

    /* initialize cfgStruct to update db cfg on single partition*/
    cfgStruct.numItems = 2;
    cfgStruct.paramArray = cfgParameters;
    cfgStruct.flags = db2CfgDatabase | db2CfgImmediate;
    cfgStruct.flags |= db2CfgSingleDbpartition;
    cfgStruct.dbname = dbAlias;
    cfgStruct.dbpartitionnum = 30;

    /* CASE 1: update database configuration */
    db2CfgSet(db2Version950, (void *)&cfgStruct, &sqlca);

    /* set cfgStruct to update db cfg on all db partitions */
    cfgStruct.flags &= ~db2CfgSingleDbpartition;

    /* CASE 2: update database configuration */
    db2CfgSet(db2Version950, (void *)&cfgStruct, &sqlca);
    .....
}
```

---

# 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****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 in the following section.

**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 in the following section.

**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 6. 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>

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

Table 6. Supported convertible data elements: snapshot variables (continued)

Snapshot variable datastream type	Structure
SQLM_ELM_LOCK_WAIT	sqlm_lockwait
SQLM_ELM_DCS_APPL	sqlm_dcs_appl, sqlm_dcs_applid_info, sqlm_dcs_appl_snap_stats, sqlm_xid, sqlm_tpmon
SQLM_ELM_DCS_DBASE	sqlm_dcs_dbase
SQLM_ELM_DCS_APPL_INFO	sqlm_dcs_applid_info
SQLM_ELM_DCS_STMT	sqlm_dcs_stmt
SQLM_ELM_COLLECTED	sqlm_collected

Table 7. 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.

---

## 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 before 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 UDB Version 7 client through a DB2 Connect Version 8 to a connected DB2 host database server.

## 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.

If a database is in the SUSPEND\_WRITE state, it cannot be put in quiesced mode.

### Authorization

One of the following authorities:

- 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\_time** database manager configuration parameter will be used.

### db2gDbQuiesceStruct data structure specific parameters

#### iDatabaseNameLen

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

## 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 where the API is run. In DB2 pureScale environments, this API can be run from any member and can trigger, when needed, a group crash recovery, which performs the crash recovery for all members of the group, or to trigger a member crash recovery.

### 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, **pParamStruct**.

#### 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). Can be NULL.

#### piTablespaceNames

Input. A pointer to a string containing a list of table space names to be dropped during the restart operation. Can 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 write operations suspended. In DB2 pureScale environments, this option resumes I/O write operations for all suspended members.

### 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.

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.

---

## 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 authorities:

- 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.

**db2gDbUnquiesceStruct data structure specific parameters**

**iDatabaseNameLen**

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

### db2DatabaseUpgrade - Upgrade previous version of DB2 database to the current release

Upgrades a DB2 database to the current release from a supported pre-DB2 Version 10.5 release.

#### Authorization

SYSADM

#### Required connection

This API establishes a database connection.

#### API include file

db2ApiDf.h

#### API and data structure syntax

```
SQL_API_RC SQL_API_FN
db2DatabaseUpgrade (
    db2UInt32    versionNumber,
    void         *pParmStruct,
    struct sqlca *pSqlca);

typedef SQL_STRUCTURE db2DatabaseUpgradeStruct
{
    char        *piDbAlias;
    char        *piUserName;
    char        *piPassword;
    db2UInt16   iDbAliasLen;
    db2UInt16   iUserNameLen;
    db2UInt16   iPasswordLen;
    db2UInt16   iOptions;
} db2DatabaseUpgradeStruct;
```

#### db2DatabaseUpgrade API parameters

##### versionNumber

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

##### pParmStruct

Input. A pointer to the db2DatabaseUpgradeStruct structure.

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

#### db2DatabaseUpgradeStruct data parameters

##### piDbAlias

Input. A string containing the alias of the database that is cataloged in the system database directory.

##### piUserName

Input. A string containing the user name of the application. May be NULL.

##### piPassword

Input. A string containing the password of the supplied user name (if any). May be NULL.

## db2DatabaseUpgrade - Upgrade previous version of DB2 database to the current release

### **iDbAliasLen**

Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

### **iPasswordLen**

Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero when no password is supplied.

### **iUserNameLen**

Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero when no user name is supplied.

### **iOptions**

Input. A bitmap indicating upgrade command options. The options are combined using the bitwise OR operator to produce a value for the parameter. Valid values (defined in `db2ApiDf.h` header file, located in the include directory) are:

#### **DB2DBUPGRADE\_REBINDALL**

Rebind all packages during database upgrade.

## **Usage notes**

This API will only upgrade a database to a higher version, and cannot be used to convert an upgraded database to its previous version.

The database must be cataloged before database upgrade.

**Important:** The support in COBOL and FORTRAN for the `db2DatabaseUpgrade` API is deprecated in Version 10.5 and might be removed in a future release. For more information, see *Support in COBOL and FORTRAN for the db2DatabaseUpgrade API* in *What's New for DB2 Version 10.5*.

---

## 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.

## 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 db2DbDirNextEntryStructV9
{
    db2UInt16 iHandle;
    struct db2DbDirInfoV9 *poDbDirEntry;
} db2DbDirNextEntryStructV9;

SQL_STRUCTURE db2DbDirInfoV9
{
    _SQLOLDCHAR alias[SQL_ALIAS_SZ];
    _SQLOLDCHAR dbname[SQL_DBNAME_SZ];
    _SQLOLDCHAR drive[SQL_DB_PATH_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 db2gDbDirNextEntryStrV9
{
    db2UInt16 iHandle;
    struct db2DbDirInfoV9 *poDbDirEntry;
} db2gDbDirNextEntryStrV9;
```

## 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 db2DbDirGetNextEntryStructV9 structure.

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

### db2DbDirNextEntryStructV9 data structure parameters

#### iHandle

Input. Identifier returned from the associated db2DbDirOpenScan API.

#### poDbDirEntry

Output. A pointer to a db2DbDirInfoV9 structure. The space for the directory data is allocated by the API, and a pointer to that space is returned to the caller.

### db2DbDirInfoV9 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).

#### SQL\_DCE

Database resides in DCE directories.



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

### **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\_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 an earlier (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**

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.

## **db2DbDirGetNextEntry - Get the next system or local database directory entry**

If db2DbDirGetNextEntry is called when there are no more entries to scan, then SQL1014N is set in the SQLCA.

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.

## 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.

## db2DbDirOpenScan - Start a system or local database directory scan

**pSqlca**

Output. A pointer to the sqlca structure.

### **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.

## **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.

## **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.

## 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 authorities:

- DATAACCESS authority
- 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/test01/lob/path, the current LOB file name is bar, and the **LOBSINSEPFIELDS** file type modifier is set, then the created LOB files will be /u/test01/LOB/path/bar.001.lob, /u/test01/LOB/path/bar.002.lob, and so on. If the **LOBSINSEPFIELDS** file



type modifier is not set, then all the LOB documents will be concatenated and put into one file /u/test01/lob/path/bar.001.lob

### **piDataDescriptor**

Input. Pointer to an `sql_dcol` structure specifying the column names for the output file. The value of the `dc_colmeth` 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 **piActionString**.

### **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 corresponding 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\_IXF**

PC version of the Integration 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 `sql_dcol` 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 export 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, 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 before completing the requested export operation, the caller action must be set to one of the following values:

#### **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/test01/xml/path`, the current XML file name is `bar`, and the **XMLINSEPF** file type modifier is set, then the created XML files will be `/u/test01/xml/path/bar.001.xml`, `/u/test01/xml/path/bar.002.xml`, and so on. If the **XMLINSEPF** file type modifier is not set, then all the XML documents will be concatenated and put into one file `/u/test01/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<sup>®</sup> servers such as DB2 for z/OS and OS/390<sup>®</sup>, DB2 for VM and VSE, and DB2 for System i<sup>®</sup>. 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 **piActionString** 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.

### 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.

**IXF** PC version of Integration Exchange Format.

**filetmod**

A host variable containing additional processing options.

**dcoldata**

A compound REXX host variable containing the column names to be used in the export file. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns (number of elements in the remainder of the variable).

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

If this parameter is NULL, or a value for **dcoldata** has not been specified, the utility uses the column names from the database table.

**msgfile**

File, path, or device name where error and warning messages are to be sent.

**number**

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

---

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

Returns the alert configuration settings for the health indicators.

**Important:** This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. It is not supported in DB2 pureScale environments. For more information, see “Health monitor has been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055045.html>.

## 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;
```

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

```
char *pPassword;  
char *pHostName;  
} db2AlertTaskAction;  
  
typedef SQL_STRUCTURE db2AlertScriptAction  
{  
    db2UInt32 scriptType;  
    db2UInt32 condition;  
    char *pPathName;  
    char *pWorkingDir;  
    char *pCmdLineParms;  
    char stmtTermChar;  
    char *pUserID;  
    char *pPassword;  
    char *pHostName;  
} db2AlertScriptAction;
```

### 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.

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

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

### **pioIndicators**

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

## **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 numericvalue 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.

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

### Usage notes

If **pioIndicators** is left NULL, all health indicators for that object will be returned. This parameter can be set to an array of db2GetAlertCfgInd structures with the **ioIndicatorID** set to the health indicator for which the configuration is wanted. 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.

---

## db2GetAlertCfgFree - Free the memory allocated by the db2GetAlertCfg API

Frees the memory allocated by the db2GetAlertCfg API.

**Important:** This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. It is not supported in DB2 pureScale environments. For more information, see “Health monitor has been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055045.html>.

### 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.

## **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 **pName** should be preallocated by the user with their corresponding 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.

## **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 **oNumGroups** = 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 corresponding maximum size.

#### **poDescription**

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

### **Usage notes**

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

---

## 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**.



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

### poContacts

Output. A pointer to the db2ContactData structure. The fields **poContacts**, **pocontacts.pAddress**, **pocontacts.pDescription**, and **pocontacts.pName** should be preallocated by the user with their corresponding 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.

---

## db2GetDistMap - Get distribution map

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 db2GetRowPartNum 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 authorities or privileges:

- DATAACCESS authority
- CONTROL privilege
- SELECT privilege

### Required connection

Database

### API include file

db2ApiDf.h

### API and data structure syntax

```
db2GetDistMap (
    sqluint32 db2VersionNumber,          /* Database version number */
    void * pParmStruct,                  /* In/out parameters */
    struct sqlca * pSqlca);              /* SQLCA */

where
SQL_STRUCTURE db2DistMapStruct
{
    unsigned char          *tname;      /* Fully qualified table */
                                   /* name */
    struct db2PartitioningInfo *partinfo; /* Partitioning */
                                   /* Information */
};

SQL_STRUCTURE db2PartitioningInfo
{
    sqluint32          pmaplen; /* Length of partitioning */
                                   /* map */
    SQL_PDB_NODE_TYPE *pmap;    /* Partitioning map */
    unsigned short      sqld;    /* # of used */
                                   /* db2PartitioningKey */
                                   /* elements */
    struct db2PartitioningKey sqlpartkey[SQL_MAX_NUM_PART_KEYS]; /* */
                                   /* KEYS */
};

SQL_STRUCTURE db2PartitioningKey
```

```
{  
    unsigned short      sqltype; /* Date Type of Key      */  
    unsigned short      sqllen;  /* Data Length of Key */  
};
```

### db2GetDistMap API parameters

**tname** The fully qualified name of the table.

**pmaplen**

The length of the distribution map.

**pmap**

The name of the distribution map.

**sqld**

The number of elements used in the **sqlpartkey** structure.

**sqlpartkey**

Distribution keys used for the table.

### Usage notes

The **pmap** in db2PartitioningInfo structure must point to an array of SQL\_PDB\_MAP\_SIZE\_32K entries.

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

Returns the list of contacts or contact groups, or both, that are notified about the health of an instance.

A contact list consists of email 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.

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

### **poContacts**

Output. 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.

## **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.

**Important:** This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. It is not supported in DB2 pureScale environments. For more information, see “Health monitor has been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055045.html>.

### **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\_RECSHEMA\_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 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 stand-alone instances, `SQLM_CURRENT_NODE` should be used.

### **iIndicatorID**

Input. The health indicator that has entered an alert state and for which a recommendation is requested. 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.

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

### **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 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.



## db2GetRecommendationsFree - Free the memory allocated by the db2GetRecommendations API

Frees the memory allocated by the db2GetRecommendations API.

**Important:** This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. It is not supported in DB2 pureScale environments. For more information, see “Health monitor has been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055045.html>.

### 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.

### 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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 previously, specify `db2Version810` or newer. 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 an earlier 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:

- `SQLM_DBMON_VERSION1`

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

- SQLM\_DBMON\_VERSION2
- SQLM\_DBMON\_VERSION5
- SQLM\_DBMON\_VERSION5\_2
- SQLM\_DBMON\_VERSION6
- SQLM\_DBMON\_VERSION7
- SQLM\_DBMON\_VERSION8
- SQLM\_DBMON\_VERSION9
- SQLM\_DBMON\_VERSION9\_5
- SQLM\_DBMON\_VERSION9\_7 (required for information about reclaiming MDC extents through a reorganization)
- SQLM\_DBMON\_VERSION10 (required for information about reclaiming ITC extents through a reorganization)

**Note:** Constants SQLM\_DBMON\_VERSION5\_2, and earlier, are deprecated and may be removed in a future release of DB2.

### **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. This field is still part of the data structure to provide compatibility with earlier releases. The functionality associated with this field is discontinued and any setting has no effect.

### **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 newer.
- node value

**Note:** For stand-alone 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 values:

- 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

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

**Note:** SQLM\_CLASS\_HEALTH and SQLM\_CLASS\_HEALTH\_WITH\_DETAIL have been deprecated and might be removed in a future release because the health monitor was deprecated in Version 9.7.

### Usage notes

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

To retrieve a health snapshot with full collection information, use the **AGENT\_ID** field in the SQLMA data structure.

## **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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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 previously, specify db2Version810 or newer. 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
- SQLM\_DBMON\_VERSION9
- SQLM\_DBMON\_VERSION9\_5

**Note:** Constants SQLM\_DBMON\_VERSION5\_2, and earlier, are deprecated and may be removed in a future release of DB2.

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

### 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 newer.
- 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 additional processing usage. 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.



## 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.

---

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

Starts HADR operations on a database.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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.

---

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

Stops HADR operations on a database.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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.

## **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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### **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.

#### DB2HADR\_FORCE\_PEERWINDOW

When this option is specified, there will not be any committed transaction loss if the command succeeds and the primary database is brought down before the end of the peer window period (set the database configuration parameter **hadr\_peer\_window** to a non-zero value). Not bringing down the primary database before the peer window expires, will result in *split brain* or *dual primary*. If executed when the HADR pair is not in a peer or disconnected peer state (the peer window has expired), an error is returned.

You cannot use this option when the synchronization mode is set to ASYNC or SUPERASYNC.

**Note:** The takeover operation with the **DB2HADR\_FORCE\_PEERWINDOW** parameter may behave incorrectly if the primary database clock and the standby database clock are not synchronized to within 5 seconds of each other. That is, the operation may succeed when it should fail, or fail when it should succeed. You should use a time synchronization service (for example, NTP) to keep the clocks synchronized to the same source.

```
/* Values for iByForce                                */
#define DB2HADR_NO_FORCE                               0  /* Do not perform START or */
                                                         /* TAKEOVER HADR operation */
                                                         /* by force                */
#define DB2HADR_FORCE                               1  /* Do perform START or    */
                                                         /* TAKEOVER HADR operation */
                                                         /* by force                */
```

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

```
#define DB2HADR_FORCE_PEERWINDOW 2    /* Perform TAKEOVER HADR    */  
                                     /* operation by force inside */  
                                     /* the Peer Window only    */
```

### 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.



## db2HistoryCloseScan - End the database history records scan

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

**Note:** This API is only supported in C, C++, or Java™ programming languages. It is no longer supported in COBOL, FORTRAN and REXX programming languages. You can issue a query to access database history records by using the DB\_HISTORY administrative view.

### 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 database history records APIs, refer to the db2HistoryOpenScan API.

---

### db2HistoryGetEntry - Get the next entry in the database history records

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

**Note:** This API is only supported in C, C++, or Java programming languages. It is no longer supported in COBOL, FORTRAN and REXX programming languages. You can issue a query to access database history records by using the DB\_HISTORY administrative view.

#### Authorization

None

#### Required connection

Instance. It is not necessary to call sqleatin 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.

## **db2HistoryGetEntry - Get the next entry in the database history records**

### **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:

#### **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 database history records APIs, refer to the db2HistoryOpenScan API.

---

## db2HistoryOpenScan - Start a database history records scan

Starts a database history records scan.

**Note:** This API is only supported in C, C++, or Java programming languages. It is no longer supported in COBOL, FORTRAN and REXX programming languages. You can issue a query to access database history records by using the DB\_HISTORY administrative view.

### Authorization

None

### Required connection

Instance. If the database is cataloged as remote, call the sqleatin 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;
    db2UInt32 oMaxLogStreams;
    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;
    db2UInt32 oMaxLogStreams;
    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 database history records entries.

#### oMaxTbspaces

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

#### oMaxLogStreams

Output. The maximum number of log streams 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 database history records.

##### **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.

## db2HistoryOpenScan - Start a database history records scan

### DB2HISTORY\_LIST\_LOAD

Lists load operations.

### DB2HISTORY\_LIST\_CRT\_TABLESPACE

Lists table space create and drop operations.

### DB2HISTORY\_LIST\_REN\_TABLESPACE

Lists table space renaming operations.

### DB2HISTORY\_LIST\_ALT\_TABLESPACE

Lists alter table space operations. The DDL field associated with an entry is not returned. To retrieve the DDL information for an entry, db2HistoryGetEntry must be called with a caller action of DB2HISTORY\_GET\_DDL immediately after the entry is fetched.

### DB2HISTORY\_LIST\_REORG

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

### oHandle

Output. Upon return from the API, this parameter contains the handle for scan access. It is subsequently used in 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 database history records.
- 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 database history records scans per process is permitted.

To list every entry in the database history records, 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 actions:

## db2HistoryOpenScan - Start a database history records scan

- Call the db2HistoryGetEntry API to fetch from the database history records.
  - 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 SQL\_RC\_NOMORE, all database history records entries have been retrieved.
  - 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.

---

## db2HistoryUpdate - Update a database history records entry

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

**Note:** This API is only supported in C, C++, or Java programming languages. It is no longer supported in COBOL, FORTRAN and REXX programming languages. You can issue a query to access database history records by using the DB\_HISTORY administrative view.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- DBADM

### Required connection

Database. To update entries in the database history records 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;
```



## db2HistoryUpdate - Update a database history records entry

```
db2UInt32 iNewLocationLen;  
db2UInt32 iNewDeviceLen;  
db2UInt32 iNewCommentLen;  
db2UInt32 iNewStatusLen;  
db2HistoryEID iEID;  
} db2gHistoryUpdateStruct;
```

### 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. Valid device types are:

D	Disk
K	Diskette
T	Tape
F	Snapshot backup
A	Tivoli Storage Manager
U	User exit
P	Pipe
N	Null device
X	XBSA
Q	SQL statement
O	Other

#### 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. Valid values are:

A	Active. The backup image is on the active log chain. Most entries are active.
---	---

## db2HistoryUpdate - Update a database history records entry

- I** Inactive. Backup images that no longer correspond to the current log sequence, also called the current log chain are flagged as inactive.
  - E** Expired. Backup images that are no longer required because there are more than **num\_db\_backups** active images are flagged as expired.
  - D** Deleted. Backup images that are no longer available for recovery should be marked as having been deleted.
  - X** Do\_not\_delete. Recovery history entries that are marked as do not delete will not be pruned or deleted by calls to the **PRUNE HISTORY** command, running the ADMIN\_CMD procedure with **PRUNE HISTORY**, calls to the db2Prune API, or automated recovery database history records pruning. You can use the do\_not\_delete status to protect key recovery file entries from being pruned and the recovery objects associated with them from being deleted.
- iEID** Input. A unique identifier that can be used to update a specific entry in the database history records.

### 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 database history records 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** parameter.

### Usage notes

This is an update function, and all information before the change is replaced and cannot be re-created. These changes are not logged.

The primary purpose of the database history records 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.

## 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 authorities:
  - DATAACCESS
  - 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 authorities:
  - DATAACCESS
  - 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 authorities:
  - DATAACCESS
  - 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 authorities:
  - 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
    - CREATEIN 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 authorities:
  - 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 authorities:
  - DATAACCESS
  - CONTROL privilege on every sub-table in the hierarchy

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

### 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 *piImportInfoOut;
    db2int32 *piNullIndicators;
    struct sqllob *piLongActionString;
} 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;

typedef SQL_STRUCTURE db2DMUXmlValidateXds
{
    struct db2Char *piDefaultSchema;
    db2UInt32 iNumIgnoreSchemas;
    struct db2Char *piIgnoreSchemas;
```

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

```
    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 sqlcol *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 sqllob *piLongActionString;
} 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;
    db2UInt64 oRowsUpdated;
    db2UInt64 oRowsRejected;
    db2UInt64 oRowsCommitted;
} db2gImportOut;
```

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

### 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 this case, the number of columns and the column specification array are both ignored. For DEL or IXF 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

Deprecated. Replaced by **piLongActionString**.

### piLongActionString

Input. Pointer to an sqllob structure containing a 4-byte long field, followed by an array of characters specifying an action that affects the table.

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]}
```

### INSERT

Adds the imported data to the table without changing the existing table data.

### INSERT\_UPDATE

Adds the imported rows if their primary key values are not in the table, and uses them for update if their primary key values are found. This option is only valid if the target table has a primary key, and the specified (or implied) list of target columns being imported includes all columns for the primary key. This option cannot be applied to views.

### REPLACE

Deletes all existing data from the table by truncating the table object, and inserts the imported data. The table definition and the index definitions are not changed. (Indexes are deleted and replaced if indexixf is in **FileTypeMod**, and **FileType** is SQL\_IXF.) If the table is not already defined, an error is returned.

**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

**Note:** The CREATE parameter is deprecated and may be removed in a future release. For additional details, see “IMPORT command options CREATE and REPLACE\_CREATE are deprecated”.

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

**Note:** The REPLACE\_CREATE parameter is deprecated and may be removed in a future release. For additional details, see “IMPORT command options CREATE and REPLACE\_CREATE are deprecated”.

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

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

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.

**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 server with a previous version of the DB2 product installed, 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.

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 Integration 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.

### **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 before completing the requested import operation, the caller action must be set to one of the following values:

#### **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

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

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 and table are being used is required. 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.

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

### 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.

### oRowsInserted

Output. Number of rows inserted into the target table.

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

### **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:

#### **- DB2DMU\_XMLVAL\_XDS**

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

## db2lImport - Import data into a table, hierarchy, nickname or view

### - 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.

## db2glImportStruct 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.

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.

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

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, perform the following actions:

1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to re-create 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.

Non-default values for **piDataDescriptor**, or specifying an explicit list of table columns in **piLongActionString**, makes importing to a remote database slower.

The database table or hierarchy must exist before data in the ASC or DEL 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.

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

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, a created 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.
- Importing bad format PC/IXF 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 want to use when doing an import operation already exists.



---

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

Inserts data from an input file or pipe into a DB2 table.

### Authorization

The authorization ID that is used to connect to the database must hold the following authorities and privileges:

- At least one of the following items:
  - DATAACCESS authority
  - CONTROL privilege on the target table
  - SELECT and INSERT privileges on the target table if the **INGEST** command specifies the INSERT statement, including as part of a MERGE statement
  - SELECT and UPDATE privileges on the target table if the **INGEST** command specifies the UPDATE statement, including as part of a MERGE statement
  - SELECT and DELETE privilege on the target table if the **INGEST** command specifies the DELETE statement (including as part of a MERGE statement)
  - INSERT, SELECT, and DELETE privilege on the target table if the **INGEST** command specifies the REPLACE statement
- SELECT privilege on the following catalog views:
  - SYSCAT.COLUMNS
  - SYSCAT.DATATYPES
  - SYSCAT.INDEXES
  - SYSCAT.INDEXCOLUSE
  - SYSCAT.SECURITYPOLICIES (if the target table has a security label column)
  - SYSCAT.TABDEP
  - SYSCAT.TABLES
  - SYSCAT.VIEWS

**Note:** Users have these privileges by default unless the database was created with the RESTRICTIVE clause.

- EXECUTE privilege on the following procedures:
  - SYSPROC.DB\_PARTITIONS (V9.7 or earlier) or SYSPROC.DB\_MEMBERS (V9.8 or later)
  - SYSPROC.MON\_GET\_CONNECTION (V9.7 and later)
- If the target table has any triggers, the authorization ID must have sufficient privileges to execute the operations that the triggers specify.
- To insert into or update a table that has protected columns, the authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise, the command fails and an error is returned.
- If an UPDATE or MERGE statement requires reading a protected column, the authorization ID must have LBAC credentials that allow read access to the column. Otherwise, the command fails and an error is returned.
- To insert into or update a table that has protected rows, the authorization ID must hold an LBAC credential that meets these criteria:
  - The LBAC credential is part of the security policy protecting the table.
  - If the security policy was defined as RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL, then the LBAC credential must be granted to the authorization ID for write access



## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

The security label on the row to be inserted, the authorization ID's LBAC credentials, the security policy definition, and the LBAC rules determine whether insert or update can be performed on the table with protected rows.

- If the **INGEST** command specifies the RESTART NEW (the default) or RESTART CONTINUE option, then SELECT, INSERT, UPDATE, and DELETE privileges are required on the restart table.
- If the **INGEST** command specifies the RESTART TERMINATE option, then SELECT and DELETE privileges are required on the restart table.
- If the **INGEST** command specifies the EXCEPTION TABLE option, then INSERT privileges are required on the exception table.

In addition, the SQL statement on the **INGEST** command is subject to the same row and column access control (RCAC) that it would be if you accessed the table without using the ingest utility.

### Required connection

Database

### API include file

db2ApiDf.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN db2Ingest (
    db2UInt32 versionNumber,
    void *pIngestStruct,
    struct sqlca *pSqlca);
```

```
typedef SQL_STRUCTURE db2IngestCfgParam
{
    db2UInt32 iCfgParam;
    db2UInt32 iCfgParamValue;
} db2IngestCfgParam;
```

```
typedef SQL_STRUCTURE db2IngestCfgList
{
    db2IngestCfgParam *piCfgParam;
    db2UInt32 iNumCfgParams;
} db2IngestCfgList;
```

```
typedef SQL_STRUCTURE db2IngestFieldDefn
{
    db2Char *piDatetimeFormat;
    db2Char *piName;
    db2UInt32 iStartPos;
    db2UInt32 iEndPos;
    db2UInt32 iLength;
    db2UInt32 iExternal;
    db2UInt32 iDecimalFormat;
    db2UInt32 iScale;
    db2UInt32 iPrecision;
    db2UInt32 iSecLabelFormat;
    db2UInt32 iEndian;
    db2UInt32 iForBitData;
    db2UInt32 iTrim;
    db2UInt32 iDefaultIfPos;
    db2UInt16 iType;
    char      iDefaultIfChar;
```

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

```
char      iEnclosedBy;
char      iRadixPoint;
} db2IngestFieldDefn;

typedef SQL_STRUCTURE db2IngestFormat
{
struct db2IngestFieldDefn *piFieldDefn;
db2Char *piFileType;
db2UInt32 iRecordLength;
db2UInt32 iNumFields;
db2UInt16 iCodepage;
db2UInt16 iImplicitlyHidden;
char iDelimiter;
} db2IngestFormat;

typedef SQL_STRUCTURE db2IngestOut
{
db2UInt64 oRowsRead;
db2UInt64 oRowsSkipped;
db2UInt64 oRowsInserted;
db2UInt64 oRowsUpdated;
db2UInt64 oRowsDeleted;
db2UInt64 oRowsMerged;
db2UInt64 oRowsRejected;
db2UInt64 oNumErrors;
db2UInt64 oNumWarnings;
db2UInt64 oMaxMsgSeverity;
} db2IngestOut;

typedef SQL_STRUCTURE db2IngestStruct
{
struct db2IngestCfgList *piCfgList;
struct sqlu_media_list *piSourceList;
struct db2IngestFormat *piFormat;
struct db2IngestOut *poIngestInfoOut;
db2Char *piDumpFile;
db2Char *piExceptTableName;
db2Char *piMsgFileName;
db2Char *piJobId;
db2Char *piSqlStatement;
db2UInt32 iWarningcount;
db2UInt32 iRestartMode;
} db2IngestStruct;

/* Possible values for field "iCfgParam". */

#define DB2INGEST_CFG_COMMIT_COUNT    1
#define DB2INGEST_CFG_COMMIT_PERIOD  2
#define DB2INGEST_CFG_NUM_FLUSHES_PER 3
#define DB2INGEST_CFG_NUM_FORMATTERS  4
#define DB2INGEST_CFG_PIPE_TIMEOUT    5
#define DB2INGEST_CFG_RETRY_COUNT      6
#define DB2INGEST_CFG_RETRY_PERIOD     7
#define DB2INGEST_CFG_SHM_MAX_SIZE     8

/* Possible values for field "iSecLabelFormat". */
#define DB2INGEST_SEC_FORMAT_NOT_SEC  0
#define DB2INGEST_SEC_FORMAT_ENCODED 1
#define DB2INGEST_SEC_FORMAT_NAME     2
#define DB2INGEST_SEC_FORMAT_STRING   3
```

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

```
/* Possible values for field "iDecimalFormat". */
#define DB2INGEST_DEC_FORMAT_DEFAULT 0
#define DB2INGEST_DEC_FORMAT_PACKED 1
#define DB2INGEST_DEC_FORMAT_ZONED 2

/* Possible values for field "iEndian". */
#define DB2INGEST_ENDIAN_DEFAULT 0
#define DB2INGEST_ENDIAN_LITTLE 1
#define DB2INGEST_ENDIAN_BIG 2

/* Possible values for field "iTrim". */

#define DB2INGEST_TRIM_DEFAULT 0
#define DB2INGEST_TRIM_NO 1
#define DB2INGEST_TRIM_LEFT 2
#define DB2INGEST_TRIM_RIGHT 3
#define DB2INGEST_TRIM 4

/* Possible values for field "iImplicitlyHidden". */
#define DB2INGEST_IMPLICIT_HID_DEFAULT 0
#define DB2INGEST_IMPLICIT_HID_MISSING 1
#define DB2INGEST_IMPLICIT_HID_INCLUDE 2

/* Possible values for field "oMaxMsgSeverity". */
#define DB2INGEST_MSGSEV_UNDEFINED 0
#define DB2INGEST_MSGSEV_INFO 1
#define DB2INGEST_MSGSEV_NO_DATA 2
#define DB2INGEST_MSGSEV_WARNING 3
#define DB2INGEST_MSGSEV_ERROR 4
#define DB2INGEST_MSGSEV_SEVERE 5

/* Possible values for field "iRestartMode". */

#define DB2INGEST_RESTART_DEFAULT 0
#define DB2INGEST_RESTART_OFF 1
#define DB2INGEST_RESTART_NEW 2
#define DB2INGEST_RESTART_CONTINUE 3
#define DB2INGEST_RESTART_TERMINATE 4
```

## db2Ingest API Parameters

### versionNumber

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

### pIngestStruct

Input. A pointer to the db2IngestStruct structure.

### pSqlca

Output. A pointer to the sqlca structure.

## db2IngestCfgParam data structure parameters

### iCfgParam

Input. Specifies the configuration parameter. The same configuration parameter cannot be set more than once per invocation. The possible values are one of the following constants:

#### DB2INGEST\_CFG\_COMMIT\_COUNT

specifies the number of rows each flusher writes in a single transaction before issuing a commit.

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### DB2INGEST\_CFG\_COMMIT\_PERIOD

Specifies the number of seconds between committed transactions.

### DB2INGEST\_CFG\_NUM\_FLUSHERS\_PER

Specifies the number of flushers to allocate for each database partition.

### DB2INGEST\_CFG\_NUM\_FORMATTERS

Specifies the number of formatters to allocate.

### DB2INGEST\_CFG\_PIPE\_TIMEOUT

specifies the maximum number of seconds to wait for data when the input source is a pipe.

### DB2INGEST\_CFG\_RETRY\_COUNT

Specifies the number of times to retry a failed, but recoverable, transaction.

### DB2INGEST\_CFG\_RETRY\_PERIOD

Specifies the number of seconds to wait before retrying a failed, but recoverable, transaction.

### DB2INGEST\_CFG\_SHM\_MAX\_SIZE

Specifies the maximum size of Inter Process Communication (IPC) shared memory in bytes on the client machine.

### iCfgParamValue

Input. Specifies the value of the configuration parameter. The possible values must be between 1 and 8, inclusive.

## db2IngestCfgList data structure parameters

### piCfgParam

Input. Pointer to an array of configuration parameter settings. If **iNumCfgParams** is 0, this field must be NULL.

### iNumCfgParams

Input. The number of elements in the array that **piCfgParam** points to.

## db2IngestFieldDefn data structure parameters

### piDatetimeFormat

Input. Specifies the format string when the field type is date, time or timestamp. If **piDatetimeFormat** is NULL then the format will be set to the default format string. This parameter must be set to NULL for all other field types.

### piName

Input. Specifies the field name. The field names must start with a dollar sign and can be 2 to 129 bytes in length (including the dollar sign) and follow the same rules as SQL identifiers. Multiplication, addition, subtraction, division, logical operator functions and user functions that can be specified in **piSqlStatement** are permitted. You can also specify delimited field names by specifying a dollar sign followed by a delimited name, for example, "\$My Field Name". This parameter cannot be NULL.

### iStartPos

Input. Specifies the field starting position. If the file type is not POSITIONAL, this parameter must be set to 0 .

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### iEndPos

Input. Specifies the field ending position. If the file type is not POSITIONAL, this parameter must be set to 0.

### iLength

Input. Specifies the field length. This length is same as the value of the SQLLEN field in the SQLDA for DATE, TIME, INTEGER, BIGINT, SMALLINT, DOUBLE. To specify the default length, set this parameter to 0. If you specify this field, it must be valid for the data type.

### iExternal

Input. Specifies the length after which the field will be truncated. Valid values are TRUE and FALSE.

### iDecimalFormat

Input. Specifies the format of the DECIMAL field. The possible values are:

#### DB2INGEST\_DEC\_FORMAT\_DEFAULT

Allowed only when the field type is DECIMAL. If **iExternal** is TRUE, this value must be specified.

#### DB2INGEST\_DEC\_FORMAT\_PACKED

Allowed only when the field type is DECIMAL.

#### DB2INGEST\_DEC\_FORMAT\_ZONED

Allowed only when the field type is DECIMAL.

**iScale** Input. Specifies the scale of a DECIMAL field. The valid values must be between 1 and SQL\_MAXDECIMAL inclusive.

### iPrecision

Input. Specifies the precision of DECIMAL, DECFLOAT, FLOAT or TIMESTAMP in digits.

- The valid values for a DECFLOAT field are 0, SQL\_DECFLOAT16\_PRECISION or SQL\_DECFLOAT34\_PRECISION
- The possible values for a TIMESTAMP field are SQL\_STAMP\_MIN\_PREC to SQL\_STAMP\_MAX\_PREC.
- The possible values for a FLOAT field are SQL\_MINSFLOATPREC to SQL\_MAXFLOATPREC.
- The possible values for a 4-byte FLOAT field are SQL\_MINSFLOATPREC to SQL\_MAXSFLOATPREC.

### iSecLabelFormat

Input. Specifies the format of a DB2SECURITYLABEL field. For more information on DB2SECURITYLABEL, refer to the **INGEST Command** documentation. The **iType** for this field is SQL\_TYP\_CHAR and the possible values are:

#### DB2INGEST\_SEC\_FORMAT\_ENCODED

Allowed when the field type is DB2SECURITYLABEL. If neither **DB2INGEST\_SEC\_FORMAT\_NAME** nor **DB2INGEST\_SEC\_FORMAT\_STRING** is specified, the default format is encoded numeric format. This is allowed only when the format is POSITIONAL.

#### DB2INGEST\_SEC\_FORMAT\_NAME

Allowed when the field type is DB2SECURITYLABEL. The DB2 security label is specified by its name. If the format is DELIMITED, either **DB2INGEST\_SEC\_FORMAT\_NAME** or **DB2INGEST\_SEC\_FORMAT\_STRING** must be specified. If this parameter is specified on any

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

DB2SECURITYLABEL field, it must have the same value on all other DB2SECURITYLABEL fields.

### **DB2INGEST\_SEC\_FORMAT\_STRING**

Allowed when the field type is DB2SECURITYLABEL. The DB2 security label is specified in string format. If the format is DELIMITED, either **DB2INGEST\_SEC\_FORMAT\_NAME** or **DB2INGEST\_SEC\_FORMAT\_STRING** must be specified. If this parameter is specified on any DB2SECURITYLABEL field, it must have the same value on all other DB2SECURITYLABEL fields.

### **DB2INGEST\_SEC\_FORMAT\_NOT\_SEC**

Specified when the field is not a DB2SECURITYLABEL.

### **iEndian**

Input. Specifies the endian format when the field type is binary numeric but not decimal. If you use a different field type or if the field format is EXTERNAL, you must set this parameter to 0. The possible values are:

#### **DB2INGEST\_ENDIAN\_DEFAULT**

The default format on the current hardware platform where the ingest utility is running.

#### **DB2INGEST\_ENDIAN\_LITTLE**

Specifies the field in little endian format (least significant byte at low address).

#### **DB2INGEST\_ENDIAN\_BIG**

Specifies the field in big endian format (most significant byte at low address).

### **iForBitData**

Input. Specifies if codepage conversion should be executed or not. Valid values are TRUE or FALSE. If the field type is CHAR, this parameter must be set to the value of TRUE to skip codepage conversion.

**iTrim** Specifies how the leading blanks, trailing blanks or both leading and trailing blanks are handled in a string. When the field type is CHAR the possible values are:

#### **DB2INGEST\_TRIM\_DEFAULT**

The default format when the format is DELIMITED. Specifies that both leading and trailing blanks in the field that are not enclosed by the string delimiter are not part of the string.

#### **DB2INGEST\_TRIM\_NO**

Specifies that the leading and trailing blanks are ignored. This is the default value when the format is POSITIONAL.

#### **DB2INGEST\_TRIM\_LEFT**

Specifies that leading blanks in the field that are not enclosed by the string delimiter are not part of the string.

#### **DB2INGEST\_TRIM\_RIGHT**

Specifies that trailing blanks in the field that are not enclosed by the string delimiter are not part of the string.

#### **DB2INGEST\_TRIM**

Specifies that both leading and trailing blanks in the field that are not enclosed by the string delimiter are not part of the string.

For other field types, the value of this parameter must be set to 0.

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### iDefaultIfPos

Specifies the position in the input record when a DEFAULTIF character is specified. If the DEFAULTIF character is not specified or if the format is DELIMITED this parameter must be set to 0.

**iType** Specifies the field type. The defined types are in the `sql.h` include file, located in the `include` subdirectory of the `sqllib` directory.

### iDefaultIfChar

Specifies the DEFAULTIF character. If you do not specify a DEFAULTIF character, this parameter must be set to 0 or '\0' (null character).

### iEnclosedBy

Specifies the character that encloses the data in the input source when the field type is CHAR, DATE, TIME, or TIMESTAMP. If there is no enclosing character, set this parameter to 0 or '\0' (null character). For all other field types, set this parameter to 0 or '\0' (null character).

### iRadixPoint

Input. Specifies the radix point when the field type is numeric. To apply the default radix point, this parameter should be set to 0 or '\0' (null character). For all other field types, this parameter should be set to 0.

## db2IngestFormat data structure parameters

### piFieldDefn

Input. Pointer to an array of field definition structures. If you set the **iNumFields** parameter to 0, you should set the **piFieldDefn** parameter to NULL. Set this parameter to 0 to allow the user to avoid manually entering the field definition list and have the utility infer the columns.

### piFileType

Input. Specifies the format of the input source. This parameter cannot be NULL. Valid values 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\_ASC

Non-delimited ASCII.

### iRecordLength

Specifies the record length when the file type is positional. If the file type is not positional this parameter must be set to 0.

### iNumFields

Specifies the number of elements in the array that **piFieldDefn** points to. This number must be less than or equal to the maximum number of fields (1012).

### iCodepage

Specifies the codepage of the input file. To apply the default code page (DB2 Application code page), you must set this parameter to 0.

### iImplicitlyHidden

Specifies whether the input records contain implicitly hidden columns when the SQL statement is INSERT, REPLACE or MERGE. Possible values for this parameter are as follows:

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### DB2INGEST\_IMPLICIT\_HID\_DEFAULT

Specifies that the value of the **DB2\_DMU\_DEFAULT** DB2 registry variable is used. If you set the **iImplicitlyHidden** parameter to **DB2INGEST\_IMPLICIT\_HID\_DEFAULT** but do not set the registry variable and the target table contains hidden columns, the utility issues an error. If the SQL statement is not an INSERT, REPLACE, or MERGE statement or if the table columns are specified, you must set the parameter to **DB2INGEST\_IMPLICIT\_HID\_DEFAULT**.

### DB2INGEST\_IMPLICIT\_HID\_MISSING

Specifies that implicitly hidden columns are omitted from the default column list.

### DB2INGEST\_IMPLICIT\_HID\_INCLUDE

Specifies that implicitly hidden columns are included in the default column list.

### iDelimiter

Specifies the column delimiter. To set the default delimiter, this parameter must be set to 0. If the file type is **POSITIONAL**, this parameter must be set to 0.

## db2IngestOut data structure parameters

### oRowsRead

Output. The number of rows the utility read from all input sources.

### oRowsSkipped

Output. This parameter is reserved for future use and is always set to 0.

### oRowsInserted

Output. The number of rows the utility inserted into the table for an SQL INSERT statement.

### oRowsUpdated

Output. The number of rows the utility updated in the table for an SQL UPDATE statement.

### oRowsDeleted

Output. The number of rows the utility deleted from the table for a SQL DELETE statement.

### oRowsMerged

Output. The number of rows the utility merged (inserted or updated) into the table for a SQL MERGE statement.

### oRowsRejected

Output. The number of rows that were rejected.

### oNumErrors

Output. The number of error messages that the utility issued. If you specify the **retry\_count** or **reconnect\_count** ingest configuration parameter, the value of the **oNumErrors** parameter does not include errors from which the utility recovered. The utility prints such errors to the message file along with a warning message to state that the utility recovered. The warning messages are counted in the **oNumWarnings** output parameter.

### oNumWarnings

Output. The number of warning messages the utility issued.



## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### oMaxMsgSeverity

Output. The maximum severity of all the messages the utility issued.  
Possible values are:

#### DB2INGEST\_MSGSEV\_UNDEFINED

No message returned.

#### DB2INGEST\_MSGSEV\_INFO

Informational ( SQL messages with message ID SQLnnnn I).

#### DB2INGEST\_MSGSEV\_NO\_DATA

No data was retrieved, updated, or deleted (Message SQL0100W ).

#### DB2INGEST\_MSGSEV\_WARNING

A warning message other than SQL0100W.

#### DB2INGEST\_MSGSEV\_ERROR

Error messages (SQL messages with message ID SQLnnnn N).

#### DB2INGEST\_MSGSEV\_SEVERE

Severe error messages (SQL messages with message ID SQLnnnn C).

Note: If the warning message is the most severe message that the API issued, the API then sets this parameter to DB2INGEST\_MSGSEV\_WARNING.

## db2IngestStruct data structure parameters

### piCfgList

Input. Pointer to the structure containing the ingest configuration parameters.

### piSourceList

Input. Specifies the input sources. For this utility, set **sqlu\_media\_list** media type member to SQLU\_CLIENT\_LOCATION.

### piFormat

Input. Pointer to the structure containing information about the input format. This parameter cannot be NULL.

### poIngestInfoOut

Pointer to the ingest output structure. If this parameter is set to NULL, the API does not return the information in the db2IngestInfoOut structure.

### piDumpFile

Input. Pointer to the dump file name. If there is no dump file, set this parameter to NULL.

### piExceptTableName

Input. Pointer to the name of the exception table. If there is no exception table, specify NULL or the empty string ("" ). If you do not specify a schema name, the value of the CURRENT SCHEMA special register is used. The API parses the table name the same way the **INGEST** command parses it. That is, if you do not delimit the schema name or table name, the API converts it to uppercase. If the schema name or table name contains characters that are not valid in a non-delimited SQL identifier, you must enclose the name in delimiters ("" ).

### piMsgFileName

Input. Pointer to the messages file name. A message file name must be specified.

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### **piJobId**

Input. Pointer to the job ID. For the default job ID, this parameter should be set to NULL. If the restart option is OFF, this parameter must be set to NULL. If the restart option is CONTINUE or TERMINATE, this parameter cannot be NULL.

### **piSqlStatement**

Input. Pointer to the SQL statement. This parameter cannot be NULL. For information about valid SQL statements, see the information about SQL statements in the **INGEST** command documentation.

### **iWarningcount**

Specifies that the Ingest utility is to stop after the warning count number has reached.

### **iRestartMode**

Specifies the Restart option if the Ingest utility fails before completing. The valid values are:

#### **DB2INGEST\_RESTART\_DEFAULT**

The default option. SELECT, INSERT, UPDATE, and DELETE privileges must exist on the restart table. Specifies that if the INGEST command fails before completing, it can be restarted from the point of the last commit by specifying the **DB2INGEST\_RESTART\_CONTINUE** option on a later INGEST command. The parameter **piJobId** is a string of up to 128 bytes that uniquely identifies the INGEST command. The string pointed to by **piJobId** must be unique across all INGEST commands in the current database that specified the RESTART option and are not yet complete. These could be commands that are still running or that failed before completing. Once the INGEST command completes, you can reuse the **piJobId** parameter with a later INGEST command. If **piJobId** is not specified, the ingest utility generates one. Before using this option, the restart log table must have been created.

#### **DB2INGEST\_RESTART\_OFF**

No Restart. Specifies that no restart information is to be saved. If the INGEST command fails before completing, it cannot be restarted using the **DB2INGEST\_RESTART\_CONTINUE** option. If you want to rerun the command to completion, the target table must be restored to its state and the INGEST command must be rerun with the same input data.

#### **DB2INGEST\_RESTART\_NEW**

SELECT, INSERT, UPDATE, and DELETE privileges must exist on the restart table. Specifies that if the INGEST command fails before completing, it can be restarted from the point of the last commit by specifying the **DB2INGEST\_RESTART\_CONTINUE** option on a later INGEST command. The parameter **piJobId** is a string of up to 128 bytes that uniquely identifies the INGEST command. The string pointed to by **piJobId** must be unique across all INGEST commands in the current database that specified the RESTART option and are not yet complete. These could be commands that are still running or that failed before completing. Once the INGEST command completes, you can reuse the **piJobId** parameter with a later INGEST command. If **piJobId** is not specified, the ingest utility generates one. Before using this option, the restart log table must have been created.

## db2Ingest API- Ingest data from an input file or pipe into a DB2 table.

### DB2INGEST\_RESTART\_CONTINUE

SELECT, INSERT, UPDATE, and DELETE privileges must exist on the restart table. Specifies that the ingest utility is to restart a previous INGEST command that specified DB2INGEST\_RESTART\_NEW option and failed before completing. The **piJobId** string specified with DB2INGEST\_RESTART\_CONTINUE option must match the **piJobId** string specified on the previous INGEST command. This restarted command is also restartable. Once the restarted command completes, you can reuse the **piJobId** string on a later INGEST command.

### DB2INGEST\_RESTART\_TERMINATE

SELECT and DELETE privileges must exist on the restart table. Specifies that the ingest utility is to clean up the restart information for a previous INGEST command that specified the DB2INGEST\_RESTART\_NEW option and failed before completing. The string specified on this option must match the **piJobId** string specified on the previous INGEST command. This option is specified when a previous restartable INGEST command fails and you plan to never resume the job. Once this option is specified, the INGEST command that failed earlier can no longer be restarted. You can, however, reuse the **piJobId** string of a later INGEST command. Note that the data committed by the original INGEST command, before its failure will still be in the target table, unlike the TERMINATE option of the LOAD command, that rolls back the operation to the point in time at which it started.

---

## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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;
    db2UInt16 iAllNodeFlag;
```

```

    db2UInt16 iNumNodes;
    db2UInt16 iLevelObjectData;
    db2UInt16 iLevelObjectIndex;
    db2UInt16 iLevelObjectLong;
    db2UInt16 iLevelObjectLOB;
    db2UInt16 iLevelObjectBlkMap;
    db2UInt16 iLevelExtentMap;
    db2UInt16 iLevelObjectXML;
    db2UInt32 iLevelCrossObject;
} 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;
    db2UInt32 iLevelCrossObject;
} 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

#### 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 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.

When DB2INSPECT\_LVL\_XOBJ\_INXDAT\_RID is used in the **iLevelCrossObject** field, the limit value specified, or the previously mentioned DEFAULT or ALL values, represent a limit in the number of errors, instead of number of pages in error, to be reported during the online index to data consistency checking.

### 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.

When DB2INSPECT\_LVL\_XOBJ\_INXDAT\_RID is used in the **iLevelCrossObject** field, as long as the index or data object is in an error state, DB2INSPECT\_ERROR\_STATE\_ALL will be ignored if specified in this field, and the online index to data consistency checking will not be performed.

### 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.

## db2Inspect - Inspect database for architectural integrity

### DB2\_ALL\_EXCEPT

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

### iNumNodes

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

### iLevelObjectData

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

#### DB2INSPECT\_LEVEL\_NORMAL

Level is normal.

#### DB2INSPECT\_LEVEL\_LOW

Level is low.

#### DB2INSPECT\_LEVEL\_NONE

Level is none.

### iLevelObjectIndex

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

#### DB2INSPECT\_LEVEL\_NORMAL

Level is normal.

#### DB2INSPECT\_LEVEL\_LOW

Level is low.

#### DB2INSPECT\_LEVEL\_NONE

Level is none.

### iLevelObjectLong

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

#### DB2INSPECT\_LEVEL\_NORMAL

Level is normal.

#### DB2INSPECT\_LEVEL\_LOW

Level is low.

#### DB2INSPECT\_LEVEL\_NONE

Level is none.

### iLevelObjectLOB

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

#### DB2INSPECT\_LEVEL\_NORMAL

Level is normal.

#### DB2INSPECT\_LEVEL\_LOW

Level is low.

#### DB2INSPECT\_LEVEL\_NONE

Level is none.

### iLevelObjectBlkMap

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

#### DB2INSPECT\_LEVEL\_NORMAL

Level is normal.

#### DB2INSPECT\_LEVEL\_LOW

Level is low.

#### DB2INSPECT\_LEVEL\_NONE

Level is none.



### iLevelExtentMap

Input. Specifies processing level for extent map. Valid values (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.

### 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.

### iLevelCrossObject

A bit-based field used for any cross object consistency checking. Valid values are:

**DB2INSPECT\_LVL\_XOBJ\_NONE**

Online index data consistency checking will not be performed (0x00000000).

**DB2INSPECT\_LVL\_XOBJ\_INXDAT\_RID**

INDEXDATA checking is enabled on RID index (0x00000001) and will be performed with IS table lock to allow for both readers and writers.

## db2glInspectStruct 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 changes using the COMMIT statement or by rolling back changes using the ROLLBACK statement, before starting the inspect operation.

## db2Inspect - Inspect database for architectural integrity

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.

When you call the db2Inspect API, you need to specify **iLevelCrossObject** in the db2InspectStruct with a proper value. When DB2INSPECT\_LVL\_XOBJ\_NONE is used, online index data consistency checking will not be performed. To enable online index data consistency checking, DB2INSPECT\_LVL\_XOBJ\_INXDAT\_RID needs to be specified in the **iLevelCrossObject** field.

The processing of table spaces will process only the objects that reside in that table space. The exception is during an index data consistency check, when data objects can reside in other table spaces and still benefit from the checking, as long as the index objects are in the table space to be inspected. For a partitioned table, each index can reside in a different table space. Only those indexes that reside in the specified table space will benefit from the index to data checking.

## db2InstanceQuiesce - Quiesce instance

Forces all users off the instance, immediately rolls back all active transaction, and puts the instance 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 instance using the db2InstanceUnquiesce API. This API allows other users to connect to the databases within the instance without having to shut down and perform another instance start.

In this mode, only groups or users with DBADM, SYSADM, SYSMANT, or SYSCTRL authority will have access to the database and its objects.

### Authorization

One of the following authorities:

- 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;
    db2UInt32 iQOptions;
} 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;
    db2UInt32 iImmediate;
```

## db2InstanceQuiesce - Quiesce instance

```
        db2UInt32 iForce;  
        db2UInt32 iTimeout;  
        db2UInt32 iQOptions;  
    } 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 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\_time** database manager configuration parameter will be used.

#### iQOptions

Input. Specifies instance quiesce options. Valid values (defined in sqlenv header file, located in the include directory) are:

##### DB2INSQUIESCE\_RESTRICTEDACCESS

The instance is quiesced with the **RESTRICTED ACCESS** option to prevent databases being activated to do authorization checking.

**db2glnsQuiesceStruct 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**.

---

## db2InstanceStart - Start instance

Starts the local or remote DB2 instance.

### Scope

This API can be issued against a single member, list of members or globally against all members. In a DB2 pureScale environment, this API may be issued against a specific host to start the instance services on that machine.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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;
    char *piNetName;
    db2UInt32 iTblspaceType;
```

```

        db2NodeType iTblspaceNode;
        db2UInt32 iIsComputer;
        char *piComputer;
        char *piUserName;
        char *piPassword;
        db2QuiesceStartStruct iQuiesceOpts;
    } db2StartOptionsStruct;

typedef SQL_STRUCTURE db2QuiesceStartStruct
{
    db2int8 iIsQRequested;
    db2int8 iQOptions;
    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
{

```

## db2InstanceStart - Start instance

```
        db2int8 iIsQRequested;  
        db2int8 iQOptions;  
        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. Pointer to the db2InstanceStartStruct structure.

#### pSqlca

Output. 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. Pointer to the name of the remote instance.

#### piCommData

Input. Pointer to the db2DasCommData structure.

#### piStartOpts

Input. 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 member 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 member to define the DB2 environment (MPP only). This file is executed before any members are started. The default value is db2profile.



**iIsNodeNum**

Input. Indicates whether a member or CF number is specified. If specified, the start command only affects the specified member or CF.

**iNodeNum**

Input. The member or CF number.

**iOption**

Input. Specifies an action. Valid values for **OPTION** (defined in sqlenv header file, located in the include directory) are:

**SQLQ\_NONE**

Issue the normal **db2start** operation.

**SQLQ\_ADDNODE**

Issue the **ADD NODE** command. This option is not supported in a DB2 pureScale environment.

**SQLQ\_RESTART**

Issue the **RESTART DATABASE** command. This option is not supported in a DB2 pureScale environment.

**SQLQ\_RESTART\_PARALLEL**

Issue the **RESTART DATABASE** command for parallel execution. This option is not supported in a DB2 pureScale environment.

**SQLQ\_STANDALONE**

Starts the node in STANDALONE mode.

**SQLQ\_HOST**

Starts up the instance on a host.

**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, or a comma delimited list of network names.

**iTblspaceType**

Input. This parameter is used for **ADD MEMBER** operations only. Specifies the type of system temporary table space definitions to be used for the member 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 member.

**SQLQ\_TABLESPACES\_LIKE\_CATALOG**

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

## db2InstanceStart - Start instance

### **iTblspaceNode**

Input. Specifies the member number from which the system temporary table space definitions should be obtained. The member number must exist in the `db2nodes.cfg` configuration file, and is only used if the **tblspace\_type** field is set to `SQLT_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.

### **iQOptions**

Input. Specifies any instance quiesce options. Valid values (defined in `sqlenv` header file, located in the `include` directory) are:

#### **DB2INSQUIESCE\_RESTRICTEDACCESS**

The instance is started in `ADMIN MODE` with the `RESTRICTED ACCESS` option to prevent databases being activated to do authorization checking.

### **piQUserName**

Input. The quiesced username.

### **piQGrpName**

Input. The quiesced group name.

### **iIsQUserGrpDef**

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**.

**Examples**

The following is an example of starting up an instance:

```
struct sqlca sqlca;                                // sqlca to carry the sqlcode
struct db2InstanceStartStruct instanceStartStruct;
struct db2StartOptionsStruct startOptions;

instanceStartStruct.iIsRemote = FALSE;              // demo local instance
instanceStartStruct.piRemoteInstName = NULL;
instanceStartStruct.piStartOpts = &startOptions;
```

In a DB2 pureScale environment, you can set individual options depending on what type of operation you want to run against the instance. For example:

- Start the instance on a host

```
startOptions.iOption = SQLE_HOST;
startOptions.iIsHostName = TRUE;
strcpy(startOptions.piHostName, "ca1");
```

- Start a member

```
startOptions.iIsNodeNum = TRUE;
startOptions.iNodeNum = 10;
```

- Start a cluster caching facility

```
startOptions.iIsNodeNum = TRUE;
startOptions.iNodeNum = 40;
```

- Perform a global **db2start** against the instance

```
instanceStartStruct.piStartOpts = NULL;
```

Finally, you need to invoke the **db2InstanceStart** API to start the instance.

```
db2InstanceStart(db2Version1010, &instanceStartStruct, &sqlca);
```

---

## db2InstanceStop - Stop instance

Stops the local or remote DB2 instance.

### Scope

This API can be issued against a single member, list of members or, globally against all members. In a DB2 pureScale environment, this API may be issued against a specific host to stop the instance services on that machine.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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 iIsHostName;
    char *piHostName;
    db2UInt32 iStopOption;
    db2UInt32 iCallerac;
    db2Sint32 iQuiesceDeferMinutes;
} db2StopOptionsStruct;
```

```

SQL_API_RC SQL_API_FN
db2gInstanceStop (
    db2Uint32 versionNumber,
    void * pParmStruct,
    struct sqlca * pSqlca);

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

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

```

## db2InstanceStop API parameters

### versionNumber

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

### pParmStruct

Input. Pointer to the db2InstanceStopStruct structure.

**pSqlca** Output. 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. Pointer to the name of the remote instance.

### piCommData

Input. Pointer to the db2DasCommData structure.

### piStopOpts

Input. 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. If a profile for the db2InstanceStart API was specified, the same profile must be specified here.

#### iIsNodeNum

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

#### iNodeNum

Input. The member number.

#### iIsHostName

Input. Indicates whether a host name is specified.

#### piHostName

Input. Pointer to the host name.

#### iStopOption

Input. Option. Valid values are:

##### SQL\_E\_NONE

Issue the normal **db2stop** operation.

##### SQL\_E\_FORCE

Issue the **FORCE APPLICATION (ALL)** command.

##### SQL\_E\_DROP

Drop the member from the db2nodes.cfg configuration file.

##### SQL\_E\_QUIESCE

Quiesce the specified member.

##### SQL\_E\_HOST

Shuts down the instance on a host.

#### iCallerac

Input. This field is valid only for the **SQL\_E\_DROP** value of the OPTION field. Valid values are:

##### SQL\_E\_DROP

Initial call. This is the default value.

##### SQL\_E\_CONTINUE

Subsequent call. Continue processing after a prompt.

##### SQL\_E\_TERMINATE

Subsequent call. Terminate processing after a prompt.

#### iQuiesceDeferMinutes

Input. Indicates the number of minutes after which applications will be forced off the member.

### 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**.

**Examples**

The following is an example of shutting down an instance.

```
struct sqlca sqlca;                                // sqlca to carry the sqlcode
struct db2InstanceStopStruct instanceStopStruct;
struct db2StopOptionsStruct stopOptions;

instanceStopStruct.iIsRemote = FALSE;              // demo local instance
instanceStopStruct.piRemoteInstName = NULL;
instanceStopStruct.piStopOpts = &stopOptions;
```

In a DB2 pureScale environment, you can set individual options depending on what type of operation you want to run against the instance. For example:

- Shut down the instance on a host
 

```
stopOptions.iOption = SQLE_HOST;
stopOptions.iIsHostName = TRUE;
strcpy(stopOptions.piHostName, "ca1");
```
- Shut down a host with the force option
 

```
stopOptions.iOption = SQLE_HOST | SQLE_FORCE;
stopOptions.iIsHostName = TRUE;
strcpy(stopOptions.piHostName, "ca1");
```
- Shut down a member
 

```
stopOptions.iIsNodeNum = TRUE;
stopOptions.iNodeNum = 10;
```
- Shut down a member with the force option
 

```
stopOptions.iOption = SQLE_FORCE;
stopOptions.iIsNodeNum = TRUE;
stopOptions.iNodeNum = 10;
```
- Shut down a cluster caching facility
 

```
stopOptions.iIsNodeNum = TRUE;
stopOptions.iNodeNum = 40;
```
- Shut down a cluster caching facility with the force option
 

```
stopOptions.iOption = SQLE_FORCE;
stopOptions.iIsNodeNum = TRUE;
stopOptions.iNodeNum = 40;
```
- Perform a global **db2stop** against the instance
 

```
instanceStopStruct.piStopOpts = NULL;
```
- Perform a global **db2stop** with the force option
 

```
stopOptions.iOption = SQLE_FORCE;
```

Finally, you need to invoke the db2InstanceStop API to shut down the instance.

```
db2InstanceStop(db2Version1010, &instanceStopStruct, &sqlca);
```

---

## db2InstanceUnquiesce - Unquiesce instance

Unquiesce all databases in the instance.

### Authorization

One of the following authorities:

- 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.



**db2glnsUnquiesceStruct data structure specific parameters**

**iInstanceNameLen**

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

---

## 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.

#### piComment

Input. Describes the DB2 server. Any comment that helps to describe the

## db2LdapCatalogDatabase - Register the database on the LDAP server

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.

---

## 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.

## **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 * pParamStruct,
    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, **pParamStruct**.

#### **pParamStruct**

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.

---

## db2LdapRegister - Register the DB2 server on the LDAP server

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

**Note:** NetBIOS is no longer supported. SNA, including its APIs APPC, APPN, and CPI-C, is also no longer supported. If you use these protocols, you must re-catalog your nodes and databases using a supported protocol such as TCP/IP. References to these protocols should be ignored.

### 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, **pParmStruct**.

## db2LdapRegister - Register the DB2 server on the LDAP server

### pParamStruct

Input. A pointer to the db2LdapRegisterStruct structure.

### pSqlca

Output. A pointer to the sqlca structure.

## 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:

## db2LdapRegister - Register the DB2 server on the LDAP server

### 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.

### 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.



## db2LdapRegister - Register the DB2 server on the LDAP server

### piIpxAddress

Input. Specify the complete IPX address. The IPX address must be specified for IPX/SPX support.

### Usage notes

Register the DB2 server once for each protocol that the server supports each time specifying a unique node name.

If any protocol configuration parameter is specified when registering a DB2 server locally, it will override the value specified in the database manager configuration file.

Only a remote DB2 server can be registered in LDAP. The computer name and the instance name of the remote server must be specified, along with the protocol communication for the remote server.

When registering a host database server, a value of SQLF\_NT\_DCS must be specified for the **iNodeType** parameter.

### 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.

## 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.

---

# 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).

**Note:** NetBIOS is no longer supported. SNA, including its APIs APPC, APPN, and CPI-C, is also no longer supported. If you use these protocols, you must re-catalog your nodes and databases using a supported protocol such as TCP/IP. References to these protocols should be ignored.

## 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 *piIpAddress;
} db2LdapProtocolInfo;
```

## db2LdapUpdate API parameters

### versionNumber

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

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

### pParamStruct

Input. A pointer to the db2LdapUpdateStruct structure.

### pSqlca

Output. A pointer to the sqlca structure.

## 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

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

the protocol type selected. For example, if SQL\_PROTOCOL\_TCP4 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.

## **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 IBM Data Server Runtime Client to connect to the host via the gateway.

#### **piBindDN**

Input. Specifies the user's LDAP distinguished name (DN). The user's

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

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.



## db2Load - Load data into a table

Loads data into a DB2 table. Although faster than the import utility, the load utility does not support loading data at the hierarchy level or loading into a nickname.

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.

### Authorization

One of the following authorities:

- DATAACCESS
- 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.

If the FORCE option is specified, SYSADM authority is required.

**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;
```

## db2Load - Load data into a table

```
    struct sqlchar *piFileTypeMod;
    char *piLocalMsgFileName;
    char *piTempFilesPath;
    struct sqlu_media_list *piVendorSortWorkPaths;
    struct sqlu_media_list *piCopyTargetList;
    db2int32 *piNullIndicators;
    struct db2LoadIn *piLoadInfoIn;
    struct db2LoadOut *poLoadInfoOut;
    struct db2PartLoadIn *piPartLoadInfoIn;
    struct db2PartLoadOut *poPartLoadInfoOut;
    db2int16 iCallerAction;
    struct sqlu_media_list *piXmlPathList;
    struct sqllob *piLongActionString;
} 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 *piXmlParse;
    db2DMUXmlValidate *piXmlValidate;
    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;

```

## db2Load - Load data into a table

```
    db2int16 iCallerAction;
    db2UInt16 iFileTypeLen;
    db2UInt16 iLocalMsgFileLen;
    db2UInt16 iTempFilesPathLen;
    struct sqlu_media_list *piXmlPathList;
    struct sqllob *piLongActionString;
} db2gLoadStruct;

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;
    db2UInt16 *piXmlParse;
    db2DMUXmlValidate *piXmlValidate;
    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;

/* Definitions for iUsing value of db2DMUXmlValidate structure */
#define DB2DMU_XMLVAL_XDS 1 /* Use XDS */
#define DB2DMU_XMLVAL_SCHEMA 2 /* Use a specified schema */
#define DB2DMU_XMLVAL_SCHEMALOC_HINTS 3 /* Use schemaLocation hints */
```

```
#define DB2DMU_XMLVAL_ORIGSCHEMA      4      /* Use schema that document was
                                              originally validated against
                                              (load from cursor only) */
```

## 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.

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 **piFileType** parameter is set to `SQL_ASC`, the **dcollmeth** field of this structure must be set to `SQL_METH_L`. The user specifies the start and end locations for each column to be loaded.

If the file type is `SQL_DEL`, **dcollmeth** 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`, **dcollmeth** 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**

Deprecated, replaced by **piLongActionString**.

**piLongActionString**

Input. Pointer to an sqllob structure containing a 4-byte long field, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE KEEPDICTIONARY|REPLACE RESETDICTIONARY|RESTART|TERMINATE
INTO tdbname [(column_list)]
[FOR EXCEPTION e_tdbname]"
```

**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.

**tdbname**

The name of the table into which the data is to be loaded. The table cannot be a system table, a declared temporary table, or a created 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.

**FOR EXCEPTION e\_tdbname**

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.

**NORANGEEXC**

Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

### NOUNIQUEEXC

Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

### piFileType

Input. A string that indicates the format of the input data source. Supported external formats (defined in `sqlutil`) are:

#### SQL\_ASC

Non-delimited ASCII.

#### SQL\_DEL

Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

#### SQL\_IXF

PC version of the Integration Exchange Format, the preferred method for exporting data from a table so that it can be loaded later into the same table or into another database manager table.

#### SQL\_CURSOR

An SQL query. The `sqlu_media_list` structure passed in through the **piSourceList** parameter is 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.

### piTempFilePath

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 **`piDataDescriptor`** 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**

Output. 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 before completing the requested load operation, the caller action must be set to one of the following value:

**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

## db2Load - Load data into a table

condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

### SQLU\_TERMINATE

Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD\_PENDING state. This option should be specified if further processing of the data is not to be done.

### SQLU\_ABORT

Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD\_PENDING state. This option should be specified if further processing of the data is not to be done.

### SQLU\_RESTART

Restart processing.

### SQLU\_DEVICE\_TERMINATE

Terminate a single device. This option should be specified if the utility is to stop reading data from the device, but further processing of the data is to be done.

### 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.

## 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.

### 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).

### 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 TRUE, 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.

### **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 table space **iUseTablespaceName** and copied over to the original table space 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 table space as the index object.

If the shadow copy is created in the same table space 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 table space 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 **savecount** 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.

## db2Load - Load data into a table

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 required. 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.

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 create for parsing, converting and formatting records when building table objects. This parameter is designed to exploit intrapartition 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 create 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.

**Important:** Starting with Version 10.1 Fix Pack 1, `SQLU_ALLOW_READ_ACCESS` is deprecated and might be removed in a future release. For more details, see See “FP1: `ALLOW READ ACCESS` parameter of the **LOAD** command is deprecated” .

#### **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:

## db2Load - Load data into a table

### 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.

During load, distribution statistics are not collected for columns of type XML.

### 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.

### 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.

```
/* XML Validate structure */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
    db2UInt16          iUsing;      /* What to use to perform */
                                /* validation */
    struct db2DMUXmlValidateXds *piXdsArgs; /* Arguments for */
                                /* XMLVALIDATE USING XDS */
    struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for */
                                /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

## db2LoadOut data structure parameters

### oRowsRead

Output. Number of records read during the load operation.

### oRowsSkipped

Output. Number of records skipped before the load operation begins.

### oRowsLoaded

Output. Number of rows loaded into the target table.

### oRowsRejected

Output. Number of records that could not be loaded.

### oRowsDeleted

Output. Number of duplicate rows deleted.

**oRowsCommitted**

Output. The total number of processed records: the number of records loaded successfully and committed to the database, plus the number of skipped and rejected records.

**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. 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.

**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.

#### **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 roll back 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.

## db2Load - Load data into a table

### **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 RECLEN file type modifier is also specified. Possible values are TRUE and FALSE. If NULL, the value defaults to FALSE.

### **piDistfile**

Input. Name of the 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.

## **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 before 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 table states 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 before calling db2Load. It is not necessary to perform a load restart or terminate operation on such database partitions.

## db2Load - Load data into a table

### 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.

### 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.

## db2gLoadIn data structure specific parameters

### iUseTablespaceLen

Input. The length in bytes of **piUseTablespace** parameter.

### 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.

```
/* XML Validate structure */
typedef SQL_STRUCTURE db2DMUXmlValidate
{
    db2UInt16                iUsing;        /* What to use to perform */
                                           /* validation */
    struct db2DMUXmlValidateXds *piXdsArgs; /* Arguments for */
                                           /* XMLVALIDATE USING XDS */
    struct db2DMUXmlValidateSchema *piSchemaArgs; /* Arguments for */
                                           /* XMLVALIDATE USING SCHEMA */
} db2DMUXmlValidate;
```

**db2gPartLoadIn data structure specific parameters****piReserved1**

Reserved for future use.

**iHostnameLen**

Input. The length in bytes of **piHostname** parameter.

**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 required, 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 before loading. The data need not be sorted when loading into an multi-dimensionally clustered (MDC) table.

---

### 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\_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 statement 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 before 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

### 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\_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 statement to take the table out of

## db2LoadQuery - Get the status of a load operation

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 before 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.

## 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**.

### 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 - Get the status of a load operation

### 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 **piLocalMsgFileName**.

## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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;
    db2UInt32 iBufferSize;
```

## db2MonitorSwitches - Get or update the monitor switch settings

```
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 previously, 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`
- `SQLM_DBMON_VERSION9`
- `SQLM_DBMON_VERSION9_5`

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

**Note:** Constants `SQLM_DBMON_VERSION5_2`, and earlier, are deprecated and may be removed in a future release of DB2.

## db2MonitorSwitches - Get or update the monitor switch settings

### iNodeNumber

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

- SQLM\_CURRENT\_NODE
- SQLM\_ALL\_NODES
- node value

**Note:** For stand-alone instances SQLM\_CURRENT\_NODE must be used.

### poOutputFormat

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

#### 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 SQLMA\_DB2 for OBJ\_TYPE (get snapshot for database manager).

The timestamp switch is unavailable if **iVersion** is less than SQLM\_DBMON\_VERSION8.

---

# 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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**.

### pParmStruct

Input. A pointer to the db2PruneStruct structure.



## db2Prune - Delete the history file entries or log files from the active log path

**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. This value is deprecated and might be removed in a future release. For more details, see See “**PRUNE LOGFILE** command is deprecated” .

**Note:** This value is not supported in DB2 pureScale environments.

**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.

If you set the **auto\_del\_rec\_obj** database configuration parameter to ON, calling db2Prune with DB2PRUNE\_OPTION\_DELETE also causes the associated backup images and load copy images to be deleted.

**DB2PRUNE\_OPTION\_LSNSTRING**

Specify that the value of piString is an LSN, used when a caller action of DB2PRUNE\_ACTION\_LOG is specified.

**Note:** This value is not supported in DB2 pureScale environments.

### db2gPruneStruct data structure specific parameters

**iStringLen**

Input. Specifies the length in bytes of **piString**.

### Usage notes

Those entries with do\_not\_delete status will not be pruned or deleted. You can set the status of recovery history file entries to do\_not\_delete using the **UPDATE HISTORY** command, the ADMIN\_CMD with **UPDATE\_HISTORY**, or the

## db2Prune - Delete the history file entries or log files from the active log path

db2HistoryUpdate API. You can use the `do_not_delete` status to prevent key recovery history file entries from being pruned or deleted.

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.

You can prune snapshot backup database history file entries using `db2Prune`, but you cannot delete the related physical recovery objects using the `DB2PRUNE_OPTION_DELETE` parameter. The only way to delete snapshot backup object is to use the **db2acsutil** command.

### 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.

## 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.

## db2ReadLog - Read log records

Reads log records from the DB2 database logs, or queries 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** or **logarchmeth2**, or both, are not set to OFF.

### Authorization

One of the following authorities:

- 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;
    db2LRI *piStartLRI;
    db2LRI *piEndLRI;
    char *poLogBuffer;
    db2UInt32 iLogBufferSize;
    db2UInt32 iFilterOption;
    db2ReadLogInfoStruct *poReadLogInfo;
    db2LRI *piMinRequiredLRI;
} db2ReadLogStruct;

typedef SQL_STRUCTURE db2ReadLogInfoStruct
{
    db2LRI initialLRI;
    db2LRI firstReadLRI;
    db2LRI nextStartLRI;
    db2LRI firstReusedLRI;
    db2UInt32 logRecsWritten;
    db2UInt32 logBytesWritten;
    db2UInt32 timeOfLRIReuse;
    db2TimeOfLog currentTimeValue;
    db2UInt32 futureUse1;
    db2LSN oldestInflightLSN;
} db2ReadLogInfoStruct;

typedef SQL_STRUCTURE db2TimeOfLog
{
    db2UInt32 seconds;
    db2UInt32 accuracy;
} db2TimeOfLog;
```

## db2ReadLog - Read log records

```
typedef SQL_STRUCTURE db2ReadLogFilterData
{
    db2LRI          recordLRIType1;
    db2LRI          recordLRIType2;
    db2UInt32       realLogRecLen;
    db2int32        sqlcode;
} db2ReadLogFilterData;

typedef SQL_STRUCTURE db2LRI
{
    db2UInt64       lriType;
    db2UInt64       part1;
    db2UInt64       part2;
} db2LRI;

#define DB2READLOG_LRI_1 1
#define DB2READLOG_LRI_2 2
```

### 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.

#### piStartLRI

Input. The starting LRI specifies the starting point for reading the log. This value does not need to be the start of an actual log record.

#### piEndLRI

Input. The ending LRI specifies the end point for reading the log. This value must be greater than the **piStartLRI** parameter, and does not need to be the start or 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 88 bytes. Its maximum size is dependent on the size of the requested range.

- If the **iFilterOption** is ON, the db2ReadLogFilterData structure will be prefixed to each log record.

- If the **iFilterOption** is OFF, each log record in the buffer is prefixed by two 24-byte db2LRI structures.

**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 LRI range.

**DB2READLOG\_FILTER\_ON**

Reads only log records in the given LRI 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 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.

**piMinRequiredLRI**

Input. This field is for future functionality. The application should set this field to 0.

**db2ReadLogInfoStruct data structure parameters****initialLRI**

The first LRI used, or that will be used, by the database since it was activated.

**firstReadLRI**

The first LRI present in **poLogBuffer** parameter.

**nextStartLRI**

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 LRI 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.

**firstReusedLRI**

The first LRI to be reused due to a database restore or rollforward operation.

**logRecsWritten**

The number of log records written to **poLogBuffer** parameter.

**logBytesWritten**

The total number of bytes of data written to **poLogBuffer** parameter.

**timeOfLRIReuse**

The time at which the LRI represented by **firstReusedLRI** was reused. The time is the number of seconds since January 1, 1970.

**currentTimeValue**

The current time according to the database.

## db2ReadLog - Read log records

### futureUse1

This field is for future functionality. The current version of DB2 always returns a value of 0.

### oldestInflightLSN

The LSN of the oldest inflight transaction.

## 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.

## db2ReadLogFilterData data structure parameters

Output. The db2ReadLogFilterData structure holds metadata for the log record, as follows:

### recordLRI

LRI of the following log record. There are two LRIs in the db2ReadLogFilterData structure:

- recordLRIType1 LRI of the log record. This is a type 1 LRI.
- recordLRIType2 LRI of the log record. This is a type 2 LRI.

### realLogRecLen

The physical log record length in the DB2 logs.

### sqlcode

This field will be non-zero if an error occurred while trying to decompress the compressed row image in the log record. If an error occurred, it will contain an integer representing the SQL code associated with the error. For permanent errors, SQL0204N will most likely be returned. Resubmitting the API request may return with the same error. For transient errors, the SQL code returned will correspond to the cause of the error, which may or may not require a user action to rectify.

## db2LRI data structure parameters

The db2LRI structure is a Log Record Identifier structure used to describe which log records the user wants to read and which log records have been read, as follows:

### lriType

There are two LRI types in the db2LRI structure that the **lriType** can be set to:

- DB2READLOG\_LRI\_1 is used when reading sequentially through the logs. This type uses an LFS as part1 and an LSN as part2 in the db2LRI structure. You use the DB2READLOG\_LRI\_1 **lriType** in db2LRI structures to read forward through a log file.
- DB2READLOG\_LRI\_2 is used to uniquely identify the log record within the log stream to which it belongs to. This type uses the logStreamID as part1 and an LSO as part1 in the db2LRI structure. You use the DB2READLOG\_LRI\_2 **lriType** in db2LRI structures to read backwards through a log file.



**Note:** DB2READLOG\_QUERY always returns DB2READLOG\_LRI\_1 db2LRI structures. To convert to a DB2READLOG\_LRI\_2 db2LRI structure, you must get the logStreamID and LSO of the current log record. You can get the LSO of the current log record by looking at the log header file. In a non-pureScale environment, the logStreamID is the same for all records in a single database.

## Usage notes

If the requested action is to read the log, you must provide a log record identifier range and a buffer to hold the log records. This API reads the log sequentially, bounded by the requested LRI 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.

**Note:** db2ReadLog API can not be used to read logs written before a drop member operation. Attempting to read those logs will result in a SQL1273N.

To use the Asynchronous Log Reader, first query the database log for a valid starting LRI. Following the query call, the read log information structure (db2ReadLogInfoStruct) will contain a valid starting LRI (in the **initialLRI** member), to be used on a read call. The value used as the ending LRI on a read can be one of the following values:

- A value greater than initialLRI
- (0000 0000 0000 0001, FFFF FFFF FFFF FFFF, FFFF FFFF FFFF FFFF) (lriType, part1, part2) 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 LRI range are returned in the log buffer. If the **ifilterOption** option is set to DB2READLOG\_FILTER\_ON, the LRI is replaced with the db2ReadLogFilterData data structure, in the buffer. Descriptions of the various DB2 log records returned by db2ReadLog can be found in the "DB2 log records" topic.

To read the next sequential log record after the initial read, use the **nextStartLRI** field returned in the db2ReadLogStruct structure. Resubmit the call, with this new starting LRI and a valid ending LRI. 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:

## db2ReadLog - Read log records

- 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.

In a DB2 pureScale environment, the call sequence for the db2ReadLog API remains unchanged from previous versions.

To begin reading log records, the caller can specify DB2READLOG\_READ or DB2READLOG\_READ\_SINGLE. The LRI passed by the caller does not necessarily exist on a log stream, in which case the database manager returns the next log record.

The db2ReadLog API returns two LRI structures in the output buffer. The first LRI structure is used to read through the logs sequentially, and the second LRI structure identifies the log record within the log stream to which it belongs.

In a DB2 pureScale environment, the database has one log stream for each member. The db2ReadLog API supports reading all of the log streams and merging these to return the log records to the caller. This causes the data changes to be returned in the order in which they took effect.

The DB2 data server can still read the log files of inactive members or offline members. The db2ReadLog API may wait for Member Crash Recovery to complete on a member before returning with more log records. In case Member Crash Recovery takes a long time to complete, SQL1015N will be returned. Users should resubmit the db2ReadLog call in this case.

In specific situations, the commit log record time might be slightly earlier than the commit occurred in the application.

### Backward compatibility of the db2ReadLog API

Any application that passes a version of the db2ReadLogStruct, that precedes DB2 Version 9.8, to a DB2 Version 9.8 or later server will fail with SQL2032N, which indicates that the db2ReadLogStruct parameter is not valid. Due to the vast differences in log record structure and the introduction of db2LRI, old applications must be rewritten and recompiled using the new db2ReadLog structures.

DB2 clients can use the old db2ReadLogStruct from DB2 Version 9.7 and pass it to a DB2 V9.7 server, or use db2ReadLogStruct for DB2 Version 9.8 and pass it to a DB2 V9.8 or DB2 V10 server.

## db2ReadLogNoConn - Read the database logs without a database connection

Extracts log records from the DB2 database logs, or queries the Log Manager for current log state information.

Before 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;
    db2LSN *piStartLSN;
    db2LSN *piEndLSN;
    char *poLogBuffer;
    db2UInt32 iLogBufferSize;
    char *piReadLogMemPtr;
    db2ReadLogNoConnInfoStruct *poReadLogInfo;
} db2ReadLogNoConnStruct;

typedef SQL_STRUCTURE db2ReadLogNoConnInfoStruct
{
    db2LSN firstAvailableLSN;
    db2LSN firstReadLSN;
    db2LSN 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 48 bytes. Its maximum size is dependent on the size of the requested range.

Each log record in the buffer is prefixed by a eight-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 is not supported in DB2 pureScale environments. Any attempts to call it will fail with SQL1419N.

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 values:

- A value greater than the caller-specified startLSN.
- FFFF 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.

**Note:** This API does not support the formatting of compressed row images in log records, which requires a connection to the database. To do this, use the db2ReadLog API instead. Since the formatting of compressed row images is not supported, this API returns the compressed row image as is.

## **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 *piDbPartitionName;
    db2LogStreamIDType *piLogStreamNum;
    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.

**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 is specified.

**piDbPartitionName**

Input. Name of the database partition that owns the recovery logs being read. This is required if the retrieve option, mentioned previously, is specified outside of a DB2 pureScale environment.

**piLogStreamNum**

Input. Log stream number that owns the recovery logs that are being read. This is required if the retrieve option, mentioned previously, is specified in a DB2 pureScale environment.

**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.

## **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.



## 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.

In a DB2 pureScale environment, the db2Recover API can be called from any member.

### Authorization

To recover an existing database, one of the following authorities:

- SYSADM
- SYSCTRL
- SYSMAINT

To recover to a new database, one of the following authorities:

- 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;
```

## db2Recover - Restore and roll forward a database

```
SQL_PDB_NODE_TYPE *piNodeList;
db2int32 iNumNodeInfo;
char *piHistoryFile;
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 * pDB2RecoverStruct,
    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

### 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 required.

#### 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 rollforward 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.

#### 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 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.

### filenameLen

Input. Length in bytes of filename.

## db2Recover - Restore and roll forward a database

### **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**

Specified 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.

## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- DBADM
- SQLADM
- 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;
    char *pPartitionName;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
    char *pTableName;
    char *pIndexName;
    char *pPartitionName;
```

## db2Reorg - Reorganize an index or a table

```
} db2ReorgIndexesAll;

SQL_API_RC SQL_API_FN
db2gReorg (
    db2UInt32 versionNumber,
    void * pReorgStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gReorgStruct
{
    db2UInt32 reorgType;
    db2UInt32 reorgFlags;
    db2int32 nodeListFlag;
    db2UInt32 numNodes;
    SQL_PDB_NODE_TYPE *pNodeList;
    union db2gReorgObject      reorgObject;
} db2gReorgStruct;

typedef SQL_STRUCTURE db2gReorgNodes
{
    SQL_PDB_NODE_TYPE nodeNum[SQL_PDB_MAX_NUM_NODE];
} db2gReorgNodes;

union db2gReorgObject
{
    struct db2gReorgTable      tableStruct;
    struct db2gReorgIndexesAll indexesAllStruct;
};

typedef SQL_STRUCTURE db2gReorgTable
{
    db2UInt32 tableNameLen;
    char *pTableName;
    db2UInt32 orderByIndexLen;
    char *pOrderByIndex;
    db2UInt32 sysTempSpaceLen;
    char *pSysTempSpace;
    db2UInt32 longTempSpaceLen;
    char *pLongTempSpace;
    db2UInt32 partitionNameLen;
    char *pPartitionName;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
    db2UInt32 tableNameLen;
    char *pTableName;
    db2UInt32 indexNameLen;
    char *pIndexName;
    db2UInt32 partitionNameLen;
    char *pPartitionName;
} 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. This value is equivalent to the CLASSIC table clause of the REORG command.

**DB2REORG\_OBJ\_TABLE\_INPLACE**

Reorganize the table inplace.

**DB2REORG\_OBJ\_INDEXESALL**

Reorganize all indexes.

**DB2REORG\_OBJ\_INDEX**

Reorganize one index.

**DB2REORG\_RECLAIM\_EXTENTS**

Reorganize a multidimensional clustering (MDC) or insert time clustering (ITC) table to reclaim empty extents for the table space.

**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 lobbs, used when DB2REORG\_OBJ\_TABLE\_OFFLINE is specified as the **reorgType**. If DB2REORG\_RESETDICTIONARY or DB2REORG\_KEEPMONITOR option is also specified, the options apply to the XML storage object of the table in addition to the table object.

**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**. This parameter is not supported in DB2 pureScale environments.

**DB2REORG\_PAUSE\_ONLINE**

Pause an existing online reorganization, used when DB2REORG\_OBJ\_TABLE\_INPLACE is specified as the **reorgType**. This parameter is not supported in DB2 pureScale environments.

**DB2REORG\_STOP\_ONLINE**

Stop an existing online reorganization, used when DB2REORG\_OBJ\_TABLE\_INPLACE is specified as the **reorgType**. This parameter is not supported in DB2 pureScale environments.

**DB2REORG\_RESUME\_ONLINE**

Resume a paused online reorganization, used when DB2REORG\_OBJ\_TABLE\_INPLACE is specified as the **reorgType**. This parameter is not supported in DB2 pureScale environments.

## db2Reorg - Reorganize an index or a table

### **DB2REORG\_NOTRUNCATE\_ONLINE**

Do not perform table truncation, used when DB2REORG\_OBJ\_TABLE\_INPLACE is specified as the **reorgType**. This parameter is not supported in DB2 pureScale environments.

### **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**. This parameter is supported in DB2 pureScale environments only when the **DB2REORG\_INDEX\_RECLAIM\_EXTENTS**, **DB2REORG\_CLEANUP\_ALL** or the **DB2REORG\_CLEANUP\_PAGES** option is also specified.

### **DB2REORG\_ALLOW\_READ**

Allow only read access to the table. This parameter is supported in DB2 pureScale environments only when the **DB2REORG\_INDEX\_RECLAIM\_EXTENTS**, **DB2REORG\_CLEANUP\_ALL** or the **DB2REORG\_CLEANUP\_PAGES** option is also specified.

### **DB2REORG\_CLEANUP\_NONE**

This value has been deprecated in Version 10.1. It indicates that no clean up is required when the reorgType is set to DB2REORG\_OBJ\_INDEXESALL or DB2REORG\_OBJ\_INDEX. Not specifying this value has the same effect, therefore, specifying the value is redundant.

### **DB2REORG\_CLEANUP\_ALL**

Clean up the committed pseudo deleted keys and committed pseudo empty pages, used when DB2REORG\_OBJ\_INDEXESALL or DB2REORG\_OBJ\_INDEX are specified as the **reorgType**.

If DB2REORG\_CLEANUP\_ALL is specified with DB2REORG\_INDEX\_REBUILD an error, SQL2218N, is returned.

### **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**.

If DB2REORG\_CLEANUP\_PAGES is specified with DB2REORG\_INDEX\_REBUILD an error, SQL2218N, is returned.

### **DB2REORG\_CLN\_OVFL\_ONLINE**

Perform a reorganization that converts overflow records into normal records. This conversion is done when DB2\_REORG\_OBJ\_INPLACE is specified as the reorganization type.

### **DB2REORG\_CONVERT\_NONE**

This value has been deprecated in Version 10.1. In earlier releases, this value indicated that no index conversion was required when the reorgType is set to DB2REORG\_OBJ\_INDEXESALL or DB2REORG\_OBJ\_INDEX. This value has become obsolete because type-1 indexes were discontinued since Version 9.7.

### **DB2REORG\_RESET\_DICTIONARY**

If the DB2REORG\_LONGLOB option is also specified, DB2REORG\_RESETDICTIONARY applies to the XML storage object of the table also. If the COMPRESS

attribute for the table is YES then a new compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary in the object. If the COMPRESS attribute for the table is NO and the table object or the XML storage object 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\_KEEP\_DICTIONARY**

If DB2REORG\_LONGLOB keyword is also specified, DB2REORG\_KEEPPDICTIONARY applies to the table object and the XML storage object of the table. If DB2REORG\_LONGLOB is not specified, the following applies only to the table object.

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\_RESET\_DICTIONARY 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**.

**DB2REORG\_INDEX\_RECLAIM\_EXTENTS**

Reclaim extents from the index object back to the table space, used when DB2REORG\_OBJ\_INDEXESALL or DB2REORG\_OBJ\_INDEX is specified as the **reorgType**.

If DB2REORG\_INDEX\_RECLAIM\_EXTENTS is specified with DB2REORG\_INDEX\_REBUILD an error, SQL2218N, is returned.

**DB2REORG\_INDEX\_REBUILD**

Rebuild the index data, used when DB2REORG\_OBJ\_INDEXESALL or DB2REORG\_OBJ\_INDEX is specified as the **reorgType**.

If DB2REORG\_INDEX\_REBUILD is specified with DB2REORG\_INDEX\_RECLAIM\_EXTENTS, DB2REORG\_CLEANUP\_ALL, or DB2REORG\_CLEANUP\_PAGES an error, SQL2218N, is returned.

**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 (such as from ALTER TABLE ADD COLUMN) and the rows were inserted before the column was added. For a nonpartitioned 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. When the **pPartitionName** parameter is specified, the temporary table space must be able to hold the specified partition.

If this parameter is not specified for a nonpartitioned 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.

#### pPartitionName

Input. Specifies the name of the data partition to reorganize.

### 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.

**pPartitionName**

Input. Specifies the name of the data partition whose indexes are to be reorganized.

**db2gReorgTable data structure specific parameters****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**

**partitionNameLen**

Input. Specifies the length, in bytes, of **pPartitionName**.

**pPartitionName**

Input. Specifies the name of the data partition to reorganize.

**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.

**partitionNameLen**

Input. Specifies the length, in bytes, of **pPartitionName**.

**pPartitionName**

Input. Specifies the name of the data partition for the index.

**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. If the objective is to reclaim space, the **RECLAIMABLE\_SPACE** output of the **ADMIN\_GET\_INDEX\_INFO** and **ADMIN\_GET\_TAB\_INFO** functions show how much space is reclaimable, in kilobytes. You can then use the **RECLAIM\_EXTENTS** option of **reorgType** or **reorgFlags** to reclaim space in your tables and indexes. The **RECLAIM\_EXTENTS** option consolidates sparse extents implicitly. This consolidation leads to more space reclamation, but a longer duration for utility execution when compared to DB2 Version 10.1. 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 **sqlarbnd** 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

## db2Reorg - Reorganize an index or a table

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 rollforward recovery following a table reorganization, both data and LONG table spaces must be rollforward 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 8. Default table access chosen based on the type of reorg and table

Type of reorg that can affect the default table access: <b>reorgType</b>	Type of applicable flags that can affect the default table access: <b>reorgFlags</b>	Access mode chosen for Non-partitioned table	Access mode chosen for Partitioned table
DB2REORG_OBJ_TABLE_OFFLINE		DB2REORG_ALLOW_READ	DB2REORG_ALLOW_NONE <sup>1</sup>
DB2REORG_OBJ_TABLE_INPLACE		DB2REORG_ALLOW_WRITE	N/A
DB2REORG_RECLAIM_EXTENTS		DB2REORG_ALLOW_WRITE	DB2REORG_ALLOW_WRITE
DB2REORG_OBJ_INDEXESALL <sup>2</sup>		DB2REORG_ALLOW_READ	DB2REORG_ALLOW_NONE <sup>1</sup>
DB2REORG_OBJ_INDEXESALL	DB2REORG_CLEANUP_ALL DB2REORG_CLEANUP_PAGES DB2REORG_INDEX_RECLAIM_EXTENTS	DB2REORG_ALLOW_READ	DB2REORG_ALLOW_READ
DB2REORG_OBJ_INDEX <sup>2</sup>	DB2REORG_CLEANUP_ALL DB2REORG_CLEANUP_PAGES DB2REORG_INDEX_REBUILD DB2REORG_INDEX_RECLAIM_EXTENTS	N/A	DB2REORG_ALLOW_READ

### Note:

1: If **pPartitionName** is not specified, DB2REORG\_ALLOW\_NONE is the default. For information about access modes when **pPartitionName** specifies a partition name, see the **REORG INDEXES/TABLE** command.

2: Unless the cleanup or reclaim reorgFlags are specified, the default is DB2REORG\_INDEX\_REBUILD.

N/A: Not applicable at this time since it is not supported.

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 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.



- If an index reorganization with allow read or allow write access is attempted on a nonpartitioned table when the indexes require rebuilding, the table is taken offline while the indexes are rebuilt. A message is written to both the administration notification log and the diagnostics log about the change. For a nonpartitioned table there is nothing to reorganize after the indexes are rebuilt. When DB2REORG\_OBJ\_INDEX is specified for a partitioned table, indexes on the table that require a rebuild are rebuilt offline. Assuming that it was not already rebuilt, the specified index is reorganized. This reorganization uses the specified access mode, and the access mode does not change during processing. A message is written to the administration notification log and the diagnostics log about the indexes being rebuilt offline.
- For classic table reorganization, if neither DB2REORG\_RESET\_DICTIONARY or DB2REORG\_KEEP\_DICTIONARY is specified, the default is DB2REORG\_KEEP\_DICTIONARY.
- If an index reorganization rebuild with no access fails, some or all indexes are not available and are rebuilt on the next table access.
- This API cannot be used with:
  - Views or an index that is based on an index extension.
  - Declared temporary tables.
  - Created temporary tables.
- With DB2 Version 9.7 Fix Pack 1 and later releases, **pPartitionName** can specify a data partition name to reorganize a specific data partition of a data partitioned table or the partitioned indexes on a specific data partition of a partitioned table. The following items apply for a data partitioned table when using **pPartitionName** to reorganize the partitioned indexes on a specific data partition of a partitioned table:
  - Only the specified data partition is restricted to the access mode level. Users are allowed to read from and write to the other partitions of the table while the partitioned indexes of a specified partition are being reorganized.
  - Only the partitioned indexes for the specified partition are reorganized. The nonpartitioned indexes on the partitioned table are not reorganized.  
If there are any nonpartitioned indexes on the table marked "invalid" or "for rebuild", all indexes marked "invalid" or "for rebuild" are rebuilt before reorganization. Otherwise, only partitioned indexes on the specified partition are reorganized or rebuilt if the index object is marked "invalid" or "for rebuild".
  - Only partitioned indexes for the specified partition are cleaned when cleaning up indexes.

When using **pPartitionName** to perform a table reorganization on a data partition of a data partitioned table, nonpartitioned indexes affect access to the table:

- If there are no nonpartitioned indexes (except system-generated XML path indexes) defined on the table, only the specified partition is reorganized. The access mode applies only to the specified partition, users are allowed to read from and write to the other partitions of the table.
- If there are nonpartitioned indexes defined on the table (excluding system-generated XML path indexes), the ALLOW NONE mode is the default and only supported access mode. In this case, the table is placed in ALLOW NONE mode. If ALLOW READ ACCESS is specified, SQL1548N is returned (SQLSTATE 5U047).
- For a data partitioned table, a table reorganization rebuilds the nonpartitioned indexes and partitioned indexes on the table after reorganizing the table. If **pPartitionName** is used to reorganize a specific data partition of a data

## db2Reorg - Reorganize an index or a table

partitioned table, a table reorganization rebuilds the nonpartitioned indexes and partitioned indexes only for the specified partition.

When using **pPartitionName** to perform a table or index reorganization, the following conditions return an error:

- If the data partition name does not exist on the given table when performing a table reorganization of a specific data partition, the reorganization fails and returns SQL2222N with reason code 1.
- If a data partition name is specified when performing an index reorganization a specific nonpartitioned index defined on a partitioned table, the reorganization fails and returns SQL2222N with reason code 2.
- If the data partition name specified is still in attached or detached state when performing a table reorganization of a data partition, the reorganization fails and returns SQL2222N with error code 3.
- If the data partition name specified is still in attached or detached state when performing an index reorganization on the partitioned indexes of a data partition, the reorganization fails and returns SQL2222N with error code 3.
- If DB2REORG\_OBJ\_INDEXESALL or DB2REORG\_OBJ\_INDEX is specified as the **reorgType** and none of the following **reorgFlags** are specified:
  - DB2REORG\_CLEANUP\_ALL
  - DB2REORG\_CLEANUP\_PAGES
  - DB2REORG\_INDEX\_RECLAIM\_EXTENTS
  - DB2REORG\_INDEX\_REBUILD

DB2REORG\_INDEX\_REBUILD is taken as the default **reorgFlag** and indexes are rebuilt.



## 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.

**Important:** This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. It is not supported in DB2 pureScale environments. For more information, see “Health monitor has been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055045.html>.

### Authorization

One of the following authorities:

- SYSADM
- SYSMANT
- 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.

## db2ResetAlertCfg - Reset the alert configuration of health indicators

### 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
- 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 table space 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.

## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- 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
- SQLM\_DBMON\_VERSION9
- SQLM\_DBMON\_VERSION9\_5

**Note:** If SQLM\_DBMON\_VERSION1 is specified as the version, the APIs cannot be run remotely.

**Note:** Constants SQLM\_DBMON\_VERSION5\_2, and earlier, are deprecated and may be removed in a future release of DB2.

#### iNodeNumber

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

- SQLM\_CURRENT\_NODE
- SQLM\_ALL\_NODES
- node value

**Note:** For stand-alone instances 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 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.

### 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), the exception being a snapshot restore where the backup image database name must be the same.
- Restore DB2 databases that were created in the two previous releases.
- Restore from a table space level backup, or restore table spaces from within a database backup image.
- Transport a set of table spaces and schemas from database backup image to a database (starting in DB2 Version 9.7 Fix Pack 2).

#### Scope

This API only affects the database partition from which it is called.

#### Authorization

To restore to an existing database, one of the following authorities:

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

To restore to a new database, one of the following authorities:

- *sysadm*
- *sysctrl*

If you specify a user name, you require CONNECT authority on the database.

#### Required connection

*Database*, to restore to an existing database. This API automatically establishes an exclusive 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.

For snapshot restore, *instance* and *database* connections are required.

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
{
    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;
    char *piStagingDBAlias;
    struct db2SchemaStruct *piSchemaList;
    char *piStogroup;
} 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;

```

## db2Restore - Restore a database or table space

```
char *piTargetDBAlias;
db2Uint32 iTargetDBAliasLen;
char *poApplicationId;
db2Uint32 iApplicationIdLen;
char *piTimestamp;
db2Uint32 iTimestampLen;
char *piTargetDBPath;
db2Uint32 iTargetDBPathLen;
char *piReportFile;
db2Uint32 iReportFileLen;
struct db2gTablespaceStruct *piTablespaceList;
struct db2gMediaListStruct *piMediaList;
char *piUsername;
db2Uint32 iUsernameLen;
char *piPassword;
db2Uint32 iPasswordLen;
char *piNewLogPath;
db2Uint32 iNewLogPathLen;
void *piVendorOptions;
db2Uint32 iVendorOptionsSize;
db2Uint32 iParallelism;
db2Uint32 iBufferSize;
db2Uint32 iNumBuffers;
db2Uint32 iCallerAction;
db2Uint32 iOptions;
char *piComprLibrary;
db2Uint32 iComprLibraryLen;
void *piComprOptions;
db2Uint32 iComprOptionsSize;
char *piLogTarget;
db2Uint32 iLogTargetLen;
struct db2gStoragePathsStruct *piStoragePaths;
char *piRedirectScript;
db2Uint32 iRedirectScriptLen;
struct db2gSchemaStruct *piSchemaList;
char *piStagingDBAlias;
db2Uint32 iStagingDBAliasLen;
char *piStogroup;
db2Uint32 iStogroupLen;
} 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.

#### piStagingDBAlias

Input. A string containing the staging database name to be used with the TRANSPORT option. If this parameter is NULL the staging database name will be internally generated and the database will be dropped after transport completes. If a name is provided the staging database will not be dropped.

#### 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.

**Note:** This parameter is obsolete, but still defined.

#### 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

## db2Restore - Restore a database or 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.

### **piSchemalist**

Input. The list of schemas to be transported. Used with **piTablespaceList** and to define a valid transportable set.

### **piMediaList**

Input. Source media for the backup image.

For more information, see the db2MediaListStruct structure.

### **piUsername**

Input. A string containing the user name to be used when attempting a connection to the database. 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 currently defined log path will be used. If the database seeds do not match the log path in the backup images, GLFH will be used instead. If the database seeds match, the log path of the GLFH currently on disk will be used.

- DB2RESTORE\_LOGTARGET\_DEFAULT

Restore log files from the backup image into the database's default log directory, for example: /home/dbuser/db2inst/NODE0000/SQL00001/LOGSTREAM0000

### **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 required.
- DB2RESTORE\_CONTINUE - Continue the restore after the user has performed some action requested by the utility (mount a new tape, for example).
- DB2RESTORE\_TERMINATE - Terminate the restore after the user has failed to perform some action requested by the utility.
- DB2RESTORE\_DEVICE\_TERMINATE - Remove a particular device from the list of devices used by restore. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action to remove the device which generated the warning from the list of devices being used.
- DB2RESTORE\_PARM\_CHK - Used to validate parameters without performing a restore. This option does not terminate the database connection after the call returns. After 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

## db2Restore - Restore a database or table space

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\_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 rollforward 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.
- DB2RESTORE\_TRANSPORT - Transport a set of table spaces and SQL schemas. The table spaces and schemas listed in the **piTablespaceList** and **piSchemaList** parameters must define a valid transportable set (starting in DB2 Version 9.7 Fix Pack 2).

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 rebuild actions listed previously.

### **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 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.

For non-snapshot restores:

- DB2RESTORE\_LOGTARGET\_DEFAULT  
Restore log files from the backup image into the database's default log directory, for example: /home/dbuser/db2inst/NODE0000/SQL00001/LOGSTREAM0000

For snapshot restore, one of the following must be given:

- DB2RESTORE\_LOGTARGET\_INCLUDE "INCLUDE"  
Restore log directory volumes from the snapshot image. If this option is specified and the backup image contains log directories, then they will be restored. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If existing log directories on disk conflict with the log directories in the backup image, then an error will be returned.
- DB2RESTORE\_LOGTARGET\_EXCLUDE "EXCLUDE"  
Do not restore log directory volumes. If this option is specified, then log directories will not be restored from the backup image. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If a path belonging to the database is restored and a log directory will implicitly be restored because of this, thus causing a log directory to be overwritten, an error will be returned.
- DB2RESTORE\_LOGTARGET\_INCFORCE "INCLUDE FORCE"  
Allow existing log directories to be overwritten and replaced when restoring the snapshot image. If this option is specified and the backup

## db2Restore - Restore a database or table space

image contains log directories, then they will be restored. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If existing log directories on disk conflict with the log directories in the backup image, then they will be overwritten by those from the backup image.

- **DB2RESTORE\_LOGTARGET\_EXCFORCE** "EXCLUDE FORCE"

Allow existing log directories to be overwritten and replaced when restoring the snapshot image. If this option is specified, then log directories will not be restored from the backup image. Existing log directories and log files on disk will be left intact if they do not conflict with the log directories in the backup image. If a path belonging to the database is restored and a log directory will implicitly be restored because of this, thus causing a log directory to be overwritten, the restore will go ahead and overwrite the conflicting log directory.

where **DB2RESTORE\_LOGTARGET\_EXCLUDE** is the default.

### **piStoragePaths**

Input. A structure containing fields that describe a list of storage paths used for automatic storage. Set this to NULL if no storage groups are created.

### **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.

### **piStogroup**

Input. The name of the target storage group for all table spaces being transported. This storage group must already exist in the target database. Only valid during a schema transport operation.

## **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: 'L'**

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

**SQLU\_XBSA\_MEDIA: 'X'**

XBSA interface.

**SQLU\_TSM\_MEDIA: 'A'**

Tivoli Storage Manager.

**SQLU\_OTHER\_MEDIA: '0'**

Vendor library.

**SQLU\_SNAPSHOT\_MEDIA: 'F'**

Specifies that the data is to be restored from a plug-in library snapshot backup.

You cannot use **SQLU\_SNAPSHOT\_MEDIA** with any of the following options:

- caller actions: **DB2RESTORE\_RESTORE\_STORDEF**, **DB2RESTORE\_STORDEF\_NOINTERRUPT**, **DB2RESTORE\_TERMINATE\_INCRE**
- **DB2RESTORE\_REPLACE\_HISTORY**
- **DB2RESTORE\_TABLESPACE**
- **DB2RESTORE\_COMPR\_LIB**
- **DB2RESTORE\_INCREMENTAL**
- **DB2RESTORE\_HISTORY**
- **DB2RESTORE\_LOGS**
- **piStoragePaths** - it must be NULL or empty in order to use it
- **piTargetDBPath**
- **piTargetDBAlias**
- **piNewLogPath**
- **iNumBuffers**
- **iBufferSize**
- **piRedirectScript**
- **iRedirectScriptLen**
- **iParallelism**
- **piComprLibrary**, **iComprLibraryLen**, **piComprOptions**, or **iComprOptionsSize**
- **numLocations** field of this structure must be 1 for snapshot restore

Also, you cannot use the **SNAPSHOT** parameter with any restore operation that involves a table space list.

The default behavior when restoring data from a snapshot backup image will be a FULL DATABASE OFFLINE restore of all paths that make up the database including all containers, local volume directory, database path (**DBPATH**), primary log and mirror log paths of the most recent snapshot backup if no timestamp is provided (**INCLUDE LOGS** is the default for all snapshot backups unless **EXCLUDE LOGS** is explicitly stated). If a timestamp is provided then that snapshot backup image will be restored.

Integrated into IBM Data Server is a DB2 ACS API driver for the following storage hardware:

- IBM TotalStorage SAN Volume Controller
- IBM Enterprise Storage Server Model 800
- IBM Storwize V7000
- IBM System Storage DS6000
- IBM System Storage DS8000



## db2Restore - Restore a database or table space

- IBM System Storage N Series
- IBM XIV

**SQLU\_SNAPSHOT\_SCRIPT\_MEDIA: 'f'**

Specifies that the data is to be restored from a scripted snapshot backup.

### 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-partition 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.

#### iStagingDBAliasLen

Input. A four-byte unsigned integer representing the length in bytes of the **piStagingDBAlias** parameter.

#### iApplicationIdLen

Input. Specifies the length in bytes of the **poApplicationId** 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 DB2RESTORE\_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.

## db2Restore - Restore a database or table space

- 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 rollforward recovery, and there is a usable current configuration file with either of these parameters enabled, following the restore, the user will be required to either take a new backup of the database, or disable the log retain and user exit parameters before connecting to the database.
- Although the restored database will not be dropped (unless restoring to a nonexistent database), if the restore fails, it will not be usable.
- If the restore type specifies that the history file 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 before 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 usable 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 DATABASE** 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 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, 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. However, if the database was created with the AUTOMATIC STORAGE NO clause, the storage paths associated with the database cannot be redefined.

### Snapshot restore

Like a traditional (non-snapshot) restore, the default behavior when restoring a snapshot backup image will be to NOT restore the log directories - DB2RESTORE\_LOGTARGET\_EXCLUDE.

If the DB2 manager detects that any log directory's group ID is shared among any of the other paths to be restored, then an error is returned. In this case, DB2RESTORE\_LOGTARGET\_INCLUDE or DB2RESTORE\_LOGTARGET\_INCFORCE must be specified, as the log directories must be part of the restore.

The DB2 manager will make all efforts to save existing log directories (primary, mirror and overflow) before the restore of the paths from the backup image takes place.

If you want the log directories to be restored and the DB2 manager detects that the preexisting log directories on disk conflict with the log directories in the backup image, then the DB2 manager will report an error. In such a case, if you have specified DB2RESTORE\_LOGTARGET\_INCFORCE, then this error will be suppressed and the log directories from the image will be restored, deleting whatever existed beforehand.

There is a special case in which the DB2RESTORE\_LOGTARGET\_EXCLUDE option is specified and a log directory path resides under the database directory (for example, /NODExxxx/SQLxxxxx/LOGSTREAMxxxxx/). In this case, a restore would still overwrite the log directory as the database path, and all of the contents beneath it, would be restored. If the DB2 manager detects this scenario and log files exist in this log directory, then an error will be reported. If you specify DB2RESTORE\_LOGTARGET\_EXCLUDE, then this error will be suppressed and those log directories from the backup image will overwrite the conflicting log directories on disk.

---

## 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, you must call this API from the catalog partition. The partitions that are rolled forward depend on what you specify in the **TO** clause:

- A point-in-time rollforward call affects all database partition servers that are listed in the `db2nodes.cfg` file.
- An **END OF LOGS** rollforward call affects the database partition servers that are specified in the **ON DATABASE PARTITION** clause. If no database partition servers are specified, the rollforward call affects all database partition servers that are listed in the `db2nodes.cfg` file.
- A database or table space rollforward call specifying end of backup affects all database partitions servers that are listed in the `db2nodes.cfg` file.

If all of the transactions on a particular database partition server have already been applied to the current database, and therefore none of those transactions need to be rolled forward, that database partition server is ignored.

When you roll forward a partitioned table to a certain point in time, you must also roll forward the table spaces that contain that table to the same point in time. However, when you roll forward a table space, you do not have to roll forward all the tables in that table space.

### Authorization

One of the following authorities:

- SYSADM
- SYSCtrl
- SYSMAINT

### Required connection

None. This API establishes an exclusive 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

```

## db2Rollforward - Roll forward a database

```
db2gRollforward (
    db2UInt32 versionNumber,
    void * pDB2gRollforwardStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
    struct db2gRfwdInputStruct *piRfwdInput;
    struct db2RfwdOutputStruct *poRfwdOutput;
} db2gRollforwardStruct;

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 rollforward 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 rollforward 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



## db2Rollforward - Roll forward a database

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.

### 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 rollforward 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 rollforward 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) rollforward 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 rollforward 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 rollforward 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 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.

#### **piDroppedTblID**

Input. A string containing the ID of the dropped table whose recovery is

## db2Rollforward - Roll forward a database

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 rollforward command.

### piExportDir

Input. The name of the directory into which the dropped table data will be exported.

### 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.

#### DB2ROLLFORWARD\_END\_OF\_BACKUP

Specifies that the database should be rolled forward to the *minimum recovery time*.

## 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 table space 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 table spaces 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.

## db2Rollforward - Roll forward a database

### 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.

### 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 before the call. This can be queried using `db2CfgGet - Get Configuration Parameters`. The `rollforward_pending` flag is set to `DATABASE` if the database is in rollforward 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 rollforward pending state when this API is called, the database will be rolled forward. Table spaces are returned to normal state after a successful database rollforward, 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 rollforward 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 rollforward 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 **lastcommit** 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 rollforward 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 rollforward 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 rollforward 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 rollforward 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. 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.

---

### db2Runstats - Update statistics for tables and indexes

Updates statistics about the characteristics of a table or any associated indexes or statistical views. This command can also update statistics about the characteristics of both a table and 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 authorities:

- SYSADM
- SYSCtrl
- SYSMaint
- DBADM
- SQLADM
- CONTROL privilege on the table
- LOAD

When used on statistical views, one of the following authorities:

- SYSADM
- SYSCtrl
- SYSMaint
- DBADM

- SQLADM
- CONTROL privilege on the view

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:

- DATAACCESS
- 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;
    double iIndexSamplingOption;
} 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;
```

## db2Runstats - Update statistics for tables and indexes

```
    db2int16 iNumFreqValues;
    db2int16 iNumQuantiles;
} db2ColumnGrpData;

SQL_API_RC SQL_API_FN
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 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 required. Each element of this array is made up of three sub-elements:

- a string that represents the name of the column on which to collect distribution statistics
- the number of frequent values to collect.
- the number of quantiles to collect

Any columns which appear in the **piColumnDistributionList** that do NOT appear in the **piColumnList**, will have basic column statistics collected on them. This would be the same effect as having included these columns in the **piColumnList** in the first place. If **iNumColDist** is zero then **piColumnDistributionList** is ignored.

**piColumnGroupList**

Input. An array of db2ColumnGrpData elements. These elements are provided when collecting column statistics on a group of columns. That is, the values in each column of the group for each row will be concatenated together and treated as a single value. Each db2ColumnGrpData is made up of 3 integer fields and an array of strings. The first integer field represents the number of strings in the array of strings **piGroupColumns**. Each string in this array contains one column name. For example, if column combinations statistics are to be collected on column groups (c1,c2) and on (c3,c4,c5) then there are 2 db2ColumnGrpData elements in **piGroupColumns**.

## db2Runstats - Update statistics for tables and indexes

The first db2ColumnGrpData element is as follows: **piGroupSize** = 2 and the array of strings contains 2 elements, namely, c1 and c2.

The second db2ColumnGrpData element is as follows: **piGroupSize** = 3 and the array of strings contains 3 elements, namely, c3, c4 and c5.

The second and the third integer fields represent the number of frequent values and the number of quantiles 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 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.

For statistical views, SYSTEM sampling can only be applied to a single base table referenced in the view definition. If the view contains multiple tables, SYSTEM sampling is possible if a single table among all the tables in the statistical view can be identified as being joined with all primary keys or unique index columns of the other tables used in the view. If the statistical view does not meet those conditions, BERNOULLI sampling will be used instead and a warning will be returned.

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 - Update statistics for tables and indexes

### 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\_UNSET\_PROFILE

Unsetting a statistics profile will remove the statistics profile from the system catalogs by setting the SYSCAT.STATISTICS\_PROFILE to NULL. If a statistics profile does not exist, attempting to unset it will result in an error (SQLCODE -2315).

### DB2RUNSTATS\_UPDATE\_PROFILE

Modify an existing statistics profile in the catalogs and collect statistics using the options from the updated profile.

### DB2RUNSTATS\_UPDA\_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.

### DB2RUNSTATS\_INDEX\_SYSTEM

Collects the index statistics on a percentage of the index pages as specified by the user via the **iIndexSamplingOption** parameter. SYSTEM sampling considers each page individually, including the index pages with probability  $P/100$  (where  $P$  is the value of **iIndexSamplingOption**) and excluding it with probability  $1-P/100$ . Thus, if **iIndexSamplingOption** 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. If DB2RUNSTATS\_INDEX\_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 **iIndexSamplingOption**) and excluding it with probability  $1-P/100$ .

### 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:

## db2Runstats - Update statistics for tables and indexes

- 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 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.

### iIndexSamplingOption

Input. Indicates that statistics are to be collected on a sample of the index data. **iIndexSamplingOption** 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 can be between 0 and 1. For example, a value of 0.01 represents one hundredth of one percent, such that one index page in 10 000 is sampled on average. A value of 0 or 100 is treated by DB2 as if sampling was not specified, regardless of whether DB2RUNSTATS\_INDEX\_SYSTEM has been specified. A value greater than 100 or less than 0 is 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\_INDEX\_SYSTEM in **iRunstatsFlags**. If the flag is not set, BERNOULLI sampling is assumed.

## db2ColumnData data structure parameters

### piColumnName

Input. Pointer to a string representing a column name.

## db2Runstats - Update statistics for tables and indexes

### 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

### 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.

For DB2 V9.7 Fix Pack 1 and later releases, the following items apply for the collection of distribution statistics for a column of type XML:

- Distribution statistics are collected for each index over XML data specified on an XML column.
- To collect distribution statistics for an index over XML data, both distribution statistics and table statistics must be collected. Table statistics must be gathered in order for distribution statistics to be collected because XML distribution statistics are stored with table statistics.

Collecting index statistics is not required to collect XML distribution statistics. Collecting index statistics without collecting distribution statistics does not collect XML distribution statistics.

By default, XML distribution statistics use a maximum of 250 quantiles.

## db2Runstats - Update statistics for tables and indexes

- Distribution statistics are collected for indexes over XML data of type VARCHAR, DOUBLE, TIMESTAMP, and DATE. Distribution statistics are not collected for indexes over XML data of type VARCHAR HASHED.
- Distribution statistics are not collected for partitioned indexes over XML data defined on a partitioned table.

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.

Statistics are not collected for columns with structured types. If they are specified, columns with these data types are ignored.

Only AVGCOLLEN and NUMNULLS are collected for columns with LOB or LONG data types.

AVGCOLLEN represents the average space in bytes when the column is stored in database memory or a temporary table. This value represents the length of the data descriptor for LOB or LONG data types, except when LOB data is inlined on the data page.

**Note:** The average space required to store the column on disk may be different than the value represented by this statistic.



## 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 DB2d11 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.

## **db2SelectDB2Copy - Select the DB2 copy to be used by your application**

### **Usage notes**

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`.

---

## 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.

---

## db2SetWriteForDB - Suspend or resume I/O writes for database

Suspends or resumes the I/O write operations for a database. I/O write operations must be suspended before a split mirror of a database can be taken.

### Scope

This API affects only the database partition where the API is run. In DB2 pureScale environments, this API can be run from any member to suspend I/O write operations for all the members, or to resume I/O write operations for all the suspended members.

### Authorization

One of the following authorities:

- 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 operations to disk.

## **db2SetWriteForDB - Suspend or resume I/O writes for database**

### **- DB2\_DB\_RESUME\_WRITE**

Resumes I/O write operations to disk.

### **piTablespaceNames**

Input. Reserved for future use.

---

## 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****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.

## db2SpmListIndTrans - List SPM indoubt transactions

### **role**

Output. Role of the Syncpoint manager.

#### **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 **oNumIndoubtsTotal**. If **oNumIndoubtsTotal** is greater than **oNumIndoubtsReturned**, the application can repeat the preceding steps.

\* To connect to the Syncpoint Manager, 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.



---

## 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.

---

### 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.

---

## 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.

---

## db2UpdateAlertCfg - Update the alert configuration settings for health indicators

Updates the alert configuration settings for health indicators.

**Important:** This command or API has been deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7. It is not supported in DB2 pureScale environments. For more information, see “Health monitor has been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055045.html>.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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;
```

## db2UpdateAlertCfg - Update the alert configuration settings for health indicators

```
    db2UInt32 iNumParmUpdates;
    struct db2AlertAttrib *piParmUpdates;
} db2AlertActionUpdate;

typedef SQL_STRUCTURE db2AlertActionDelete
{
    db2UInt32 iActionType;
    char *piName;
    db2UInt32 iCondition;
} db2AlertActionDelete;

typedef SQL_STRUCTURE db2AlertActionNew
{
    db2UInt32 iActionType;
    struct db2AlertScriptAction *piScriptAttribs;
    struct db2AlertTaskAction *piTaskAttribs;
} db2AlertActionNew;

typedef SQL_STRUCTURE db2AlertScriptAction
{
    db2UInt32 scriptType;
    db2UInt32 condition;
    char *pPathName;
    char *pWorkingDir;
    char *pCmdLineParms;
    char stmtTermChar;
    char *pUserID;
    char *pPassword;
    char *pHostName;
} db2AlertScriptAction;

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

### 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

## db2UpdateAlertCfg - Update the alert configuration settings for health indicators

- DB2ALERTCFG\_OBJTYPE\_TS\_CONTAINER

### piObjName

Input. The name of the table space or table space container when object type, **iObjType**, is set to DB2ALERTCFG\_OBJTYPE\_TABLESPACE or DB2ALERTCFG\_OBJTYPE\_TS\_CONTAINER, otherwise set to NULL.

### piDbName

Input. The alias name for the database for which configuration is requested when object type, **iObjType**, is DB2ALERTCFG\_OBJTYPE\_TS\_CONTAINER, DB2ALERTCFG\_OBJTYPE\_TABLESPACE, and DB2ALERTCFG\_OBJTYPE\_DATABASE, otherwise set to NULL.

### iIndicatorID

Input. The health indicator for which the configuration updates are to apply.

### iNumIndAttribUpdates

Input. The number of alert attributes to be updated for the **iIndicatorID** health indicator.

### piIndAttribUpdates

Input. A pointer to the db2AlertAttrib structure array.

### iNumActionUpdates

Input. The number of alert actions to be updated for the **iIndicatorID** health indicator.

### piActionUpdates

Input. A pointer to the db2AlertActionUpdate structure array.

### iNumActionDeletes

Input. The number of alert actions to be deleted from the **iIndicatorID** health indicator.

### piActionDeletes

Input. A pointer to the db2AlertActionDelete structure array.

### iNumNewActions

Input. The number of new alert actions to be added to the **iIndicatorID** health indicator.

### piNewActions

Input. A pointer to the db2AlertActionNew structure array.

## 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 numeric 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 numeric value defined in sqlmon.

### db2AlertActionNew data structure parameters

#### iActionType

Input. Specifies the alert action. Valid values are:

- DB2ALERTCFG\_ACTIONTYPE\_SCRIPT
- DB2ALERTCFG\_ACTIONTYPE\_TASK

## db2UpdateAlertCfg - Update the alert configuration settings for health indicators

### **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 numeric 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.

### **pUserID**

The user account under which the script will be executed.



## **db2UpdateAlertCfg - Update the alert configuration settings for health indicators**

### **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.

## **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 authorities:

- 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.

---

## 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.

## **db2UpdateContact - Update the attributes of a contact**

### **iNumAttribsUpdated**

Input. The number attributes to be updated.

### **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.

---

### 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.

## **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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### **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.



## **db2UpdateHealthNotificationList - Update the list of contacts to whom health alert notifications can be sent**

### **db2UpdateHealthNotificationListData data structure parameters**

#### **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.

---

## 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

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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.

---

## 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 authorizations:

- DBADM authority
- If EXPLAIN ONLY is specified, EXPLAIN authority or an authority that implicitly includes EXPLAIN is sufficient.
- If a package does not exist, BINDADD authority and:
  - If the schema name of the package does not exist, IMPLICIT\_SCHEMA authority on the database.
  - If the schema name of the package does exist, CREATEIN privilege on the schema.
- If the package exists, one of the following privileges:
  - ALTERIN privilege on the schema
  - BIND privilege on the package

In addition, if capturing explain information using the EXPLAIN or the EXPLSNAP clause, one of the following authorizations is required:

- INSERT privilege on the explain tables
- DATAACCESS authority

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.

### 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

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 might 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.

## sqlabndx - Bind application program to create a package

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called `myapp.sqc` generates a default bind file called `myapp.bnd` and a default package name of `MYAPP`. (However, the bind file name and the package name can be overridden at precompile time by using the `SQL_BIND_OPT` and the `SQL_PKG_OPT` options of `sqlaprep`.)

**BIND** executes under the transaction that the user has started. After performing the bind, **BIND** issues a `COMMIT` (if bind is successful) or a `ROLLBACK` (if bind is unsuccessful) operation to terminate the current transaction and start another one.

Binding halts if a fatal error or more than 100 errors occur. If a fatal error occurs during binding, **BIND** stops binding, attempts to close all files, and discards the package.

Binding application programs have prerequisite requirements and restrictions beyond the scope of this manual. For example, an application cannot be bound from a 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.

## 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.

If a positive line width is specified, new line sequences are inserted between words so that the lines do not exceed the line width.

## sqlaintp - Get error message

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.



## 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 authorizations:

- DBADM authority
- If EXPLAIN ONLY is specified, EXPLAIN authority or an authority that implicitly includes EXPLAIN is sufficient.
- If SQLERROR CHECK or EXPLAIN ONLY is specified, either EXPLAIN or SQLADM authority is sufficient.
- If a package does not exist, BINDADD authority and:
  - If the schema name of the package does not exist, IMPLICIT\_SCHEMA authority on the database.
  - If the schema name of the package does exist, CREATEIN privilege on the schema.
- If the package exists, one of the following privileges:
  - ALTERIN privilege on the schema
  - BIND privilege on the package

In addition, if capturing explain information using the EXPLAIN or the EXPLSNAP clause, one of the following authorizations is required:

- INSERT privilege on the explain tables
- DATAACCESS authority

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.

### Required connection

Database

### API include file

`sql.h`

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqlaprep (
    _SQLLDCHAR * pProgramName,
    _SQLLDCHAR * pMsgFileName,
    struct sqlopt * pPrepOptions,
    struct sqlca * pSqlca);
```

SQL\_API\_RC SQL\_API\_FN

## sqlaprep - Precompile application program

```
sqlgprep (  
    unsigned short MsgFileNameLen,  
    unsigned short ProgramNameLen,  
    struct sqlca * pSqlca,  
    struct sqlopt * pPrepOptions,  
    _SQLLDCHAR * pMsgFileName,  
    _SQLLDCHAR * pProgramName);
```

### sqlaprep API parameters

#### 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 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 stops precompiling, attempts to close all files, and discards the package.

The Precompile option types and values are defined in `sql.h`.

When using the **PRECOMPILE** command or sqlaprep API, the name of the package can be specified with the **PACKAGE USING** option. When using this option, up to 128 bytes may be specified for the package name. When this option is not used, the name of the package is generated by the precompiler. The name of the application program source file (minus extension and folded to uppercase) is used up to a maximum of 8 characters. The name generated will continue to have a maximum of 8 bytes to be compatible with previous versions of DB2.

### REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

---

## sqlarbnd - Rebind package

Allows the user to re-create a package stored in the database without the need for a bind file.

### Authorization

One of the following authorities:

- 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
sqlarbnd (
    char * pPackageName,
    struct sqlca * pSqlca,
    struct sqlopt * pRebindOptions);
```

```
SQL_API_RC SQL_API_FN
sqlgrbnd (
    unsigned short PackageNameLen,
    char * pPackageName,
    struct sqlca * pSqlca,
    struct sqlopt * pRebindOptions);
```

### sqlarbnd 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 re-create a package. This enables the user to take advantage of a change in the system without a need for the original bind file. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, **REBIND** can be used to re-create the package. **REBIND** can also be used to re-create packages after db2Runstats has been executed, thereby taking advantage of the new statistics.
- Provides a method to re-create 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 database upgrade, all packages stored in the database will be invalidated by the **UPGRADE DATABASE** command. 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.

## sqlarbnd - Rebind package

**REBIND** is supported by DB2 Connect.

If **REBIND** is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the SYSCAT.PACKAGES system catalog table during the rebind.

When **REBIND** is executed, the database manager re-creates 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.

---

## sqlbctcq - Close a table space container query

Ends a table space container query request and frees the associated resources.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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.

### sqlbctsq - Close a table space query

Ends a table space query request, and frees up associated resources.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

#### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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.



## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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.

## sqlbftcq - Fetch the query data for rows in a table space container

**pNumContainers**

Output. Number of rows of output returned.

### Usage notes

The user is responsible for allocating and freeing the memory pointed to by the **pContainerData** parameter. This API can only be used after a successful sqlbotcq call. It can be invoked repeatedly to fetch the list generated by sqlbotcq.

## 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.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

### Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- sysmaint
- SYSMON
- 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.

## sqlbftpq - Fetch the query data for rows in a table space

### MaxTablespaces

Input. The maximum number of rows of data that the user allocated output area (pointed to by **pTablespaceData**) can hold.

### pTablespaceData

Input and output. Pointer to the output area, a structure for query data. For more information about this structure, see SQLB-TBSPQRY-DATA. The caller of this API must:

- Allocate space for **MaxTablespaces** of these structures
- Initialize the structures
- Set TBSPQVER in the first structure to SQLB\_TBSPQRY\_DATA\_ID
- Set **pTablespaceData** to point to this space. The API will use this space to return the table space data.

### pNumTablespaces

Output. Number of rows of output returned.

## Usage notes

The user is responsible for allocating and freeing the memory pointed to by the **pTablespaceData** parameter. This API can only be used after a successful sqlbotsq call. It can be invoked repeatedly to fetch the list generated by sqlbotsq.

## sqlbgtss - Get table space usage statistics

Provides information about the space utilization of a table space.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

### Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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.

## sqlbgtss - Get table space usage statistics

### **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.

## 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.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

### Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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.

## sqlbmtsq - Get the query data for all table spaces

### **pNumTablespaces**

Output. The total number of table spaces in the connected database.

### **pppTablespaceData**

Output. The caller supplies the API with the address of a pointer. The space for the table space query data is allocated by the API, and a pointer to that space is returned to the caller. On return from the call, the pointer points to an array of SQLB\_TBSPQRY\_DATA pointers to the complete set of table space query data.

### **reserved1**

Input. Always SQLB\_RESERVED1.

### **reserved2**

Input. Always SQLB\_RESERVED2.

## **Usage notes**

This API uses the lower level services, namely:

- sqlbotsq
- sqlbftpq
- sqlbctsq

to get all of the table space query data at once.

If sufficient memory is available, this function returns the number of table spaces, and a pointer to the memory location of the table space query data. It is the user's responsibility to free this memory with a call to sqlefm.

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.



## 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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

## sqlbotcq - Open a table space container query

this information. Use this API to scan the list of containers for specific information. Using this API is identical to calling the following three APIs - sqlbotcq, sqlbftcq, and sqlbctcq, except that this API automatically allocates the memory for the output information. A call to this API must be followed by a call to sqlefmem to free the memory.

- sqlbotcq, sqlbftcq, and 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.

## sqlbotsq - Open a table space query

Prepares for a table space query operation, and returns the number of table spaces currently in the database.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMAINT
- SYSMON
- 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
sqlgothsq (
    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.

## sqlbotsq - Open a table space query

### pNumTablespaces

Output. The number of table spaces in the connected database.

### Usage notes

This API is normally followed by one or more calls to sqlbftpq, and then by one call to sqlbctsq.

An application can use the following APIs to fetch information about the currently defined table spaces:

- sqlbstpq  
Fetches information about a given table space. Only one table space entry is returned (into a space provided by the caller). Use this API when the table space identifier is known, and information about only that table space is desired.
- sqlbmtsq  
Fetches information about all table spaces. The API allocates the space required to hold the information for all table spaces, and returns a pointer to this information. Use this API to scan the list of table spaces when searching for specific information. Using this API is identical to calling the three APIs mentioned in the following list, 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.

## sqlbstpq - Get information about a single table space

Retrieves information about a single currently defined table space.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

### Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMAINT
- SYSMON
- 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.

## sqlbstpq - Get information about a single table space

### **pTablespaceData**

Input and output. Pointer to a user-supplied SQLB\_TBSPQRY\_DATA structure where the table space information will be placed upon return. The caller of this API must initialize the structure and set **TBSPQVER** to SQLB\_TBSPQRY\_DATA\_ID (in sqlutil).

### **reserved**

Input. Always SQLB\_RESERVED1.

## **Usage notes**

This API retrieves information about a single table space if the table space identifier to be queried is known. This API provides an alternative to the more expensive OPEN TABLESPACE QUERY, FETCH, and CLOSE combination of APIs, which must be used to scan for the required 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.

## sqlbstsc - Set table space containers

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 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 before the restoration of database pages.

### Authorization

One of the following authorities:

- 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**

## sqlbstsc - Set table space containers

holds. A value of 0 provided with a NULL pointer for **pContainerData** indicates that the table space is to be managed by automatic storage.

### 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. A NULL value along with a 0 value for **NumContainers** indicates that the table space is to be managed by automatic storage. This option can also be used to provide better striping for existing automatic storage enabled table spaces on the existing storage paths by redefining the containers.

**Note:** The table space will be offline while being restored.

## 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 DB2RESTORE\_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 db2Restore a second time with **CallerAction** set to DB2RESTORE\_CONTINUE.

This sequence of actions 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.

Using this API to enable automatic storage for table spaces will cause all current containers to be redefined to use the storage paths provided to the database.

Existing system-managed (SMS) table spaces cannot be converted to use automatic storage.

SetContainerOptions is ignored when a table space is being converted to use automatic storage (**NumContainers** is 0, and **pContainerData** is NULL).



A redirected restore of a table space in a multi-partition environment using the **USING AUTOMATIC STORAGE** option of **SET TABLESPACE CONTAINERS** command only converts the table space to automatic storage on the partition being restored. The containers on any other database partition are not redefined.

**Note:** Converting the table space on only one of the partitions to automatic storage as part of a redirected restore operation causes inconsistencies in the definition of the table space. Unexpected results could also be caused when adding new database partitions to the system or to the database partition group. For example, if all of the database partitions were subject to a redirected restore followed by using the **USING AUTOMATIC STORAGE** option of the **SET TABLESPACE CONTAINERS** command, then the table space will be converted to automatic storage on all the database partitions. Adding another database partition later will have the same definition for the table space as that found on the other database partitions.

---

## 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.

**Important:** This command or API has been deprecated in Version 9.7 and might be removed in a future release. You can use the MON\_GET\_TABLESPACE and the MON\_GET\_CONTAINER table functions instead which return more information. For more information, see “LIST TABLESPACES and LIST TABLESPACE CONTAINERS commands have been deprecated” at <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.wn.doc/doc/i0055001.html>.

### Scope

In a partitioned database environment, only the table spaces on the current database partition are listed.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT
- SYSMON
- 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.

## sqlbctq - Get the query data for all table space containers

### TablespaceId

Input. ID of the table space for which container data is desired, or a special ID, SQLB\_ALL\_TABLESPACES (defined in `sqlutil`), which produces a list of all containers for the entire database.

### pNumContainers

Output. The number of containers in the table space.

### ppContainerData

Output. The caller supplies the API with the address of a pointer to a `SQLB_TBSCONTQRY_DATA` structure. The space for the table space container query data is allocated by the API, and a pointer to that space is returned to the caller. On return from the call, the pointer to the `SQLB_TBSCONTQRY_DATA` structure points to the complete set of table space container query data.

## Usage notes

This API uses the lower level services, namely:

- `sqlbotcq`
- `sqlbftcq`
- `sqlbctcq`

to get all of the table space container query data at once.

If sufficient memory is available, this function returns the number of containers, and a pointer to the memory location of the table space container query data. It is the user's responsibility to free this memory with a call to `sqlefmem`. If sufficient memory is not available, this function simply returns the number of containers, and no memory is allocated. If this should happen, use `sqlbotcq`, `sqlbftcq`, and `sqlbctcq` to fetch less than the whole list at once.

---

## sqlcspqy - List DRDA indoubt transactions

Provides a list of transactions that are indoubt between the syncpoint manager partner connections.

This API is being deprecated. 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.

## sqlc\_activate\_db - Activate database

Activates the specified database and starts up all associated database services, so that the database is available for connection and use by any application.

### Scope

This API activates the specified database on all members. If one or more of these members encounters an error during activation of the database, a warning is returned. The database remains activated on all members on which the API has succeeded.

**Note:** If an error is encountered then the database may remain deactivated on those members where the API failed.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL
- SYSMANT

### 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
db2ActivateDb (
    db2UInt32 versionNumber,
    void * pDB2ActivateDbStruct,
    struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ActivateDbStruct
{
    char * piDbAlias;
    char * piUserName;
    char * piPassword;
    db2UInt32 iOptions;
} db2ActivateDbStruct;
```

### db2ActivateDb API parameters

#### versionNumber

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

#### pDB2ActivateDbStruct

Input. Pointer to the db2ActivateDbStruct structure.

**pSqlca** Output. Pointer to the sqlca structure.

## sqlc\_activate\_db - Activate database

### db2ActivateDbStruct data structure parameters

#### piDbAlias

Input. Pointer to the database alias name.

#### piUserName

Input. Pointer to the user ID starting the database. Can be NULL.

#### piPassword

Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

#### iOptions

Input. Reserved for future use.

### 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.

#### 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. Pointer to the sqlca structure.

### sqlg\_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.

## Examples

This section provides an example of a database activation scenario. The following code block is a common body program that will be used for the database activation scenarios.

```
struct sqlca sqlca;           // sqlca to carry the sqlcode
struct db2ActivateDbStruct activateDbStruct;
struct db2ActDeactMemberStruct memberList;

strcpy(activateDbStruct->piUserName, "USER1"); //local instance
```

The following activate API is shown common to the scenarios listed in the following examples:

```
db2ActivateDb( db2Version, &activateDbStruct, &sqlca );
```

- Activate a database globally across all members

```
activateDbStruct.iOptions = DB2_ACTDEACT_DB_GLOBAL;  
activateDbStruct->piMemberList = NULL;
```

- Activate a database at an individual member 20

```
activateDbStruct.iOptions = DB2_ACTDEACT_DB_MEMBER;  
activateDbStruct->piMemberList->numMembers = 1;  
activateDbStruct->piMemberList->pMember[0] = 20;
```

### Usage notes

Database administrators can use **ACTIVATE DATABASE** to start 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, you may invoke the `db2GetSnapshot` API.

If a database was started by a **DB2 CONNECT TO** command (or by an implicit connect), and subsequently an **ACTIVATE DATABASE** is issued for that same database, then **DEACTIVATE DATABASE** must also be used to shut down that database.

See the **autorestart** configuration parameter for database connection behavior when the target database is in an inconsistent state.

### REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

---

## sqlc\_deactivate\_db - Deactivate database

Deactivates the specified database and stops all associated database services.

### Scope

This API deactivates the specified database on all members. If one or more of these members encounters an error during deactivation of the database, a warning is returned. The database will be deactivated on all members where the API executed successfully.

**Note:** If an error is encountered then the database may remain activated on those members where the API failed.

### Authorization

One of the following authorities:

- 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
    db2DeactivateDb(
        db2Uint32 versionNumber,
        void * pDB2DeactivateDbStruct,
        struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DeactivateDbStruct
{
    char * piDbAlias;
    char * piUserName;
    char * piPassword;
    db2Uint32 iOptions;
    db2Uint32 iNumMembers;
    SQL_PDB_NODE_TYPE * piMemberList;
} db2DeactivateDbStruct;
```

### db2DeactivateDb API parameters

#### versionNumber

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

#### pDB2DeactivateDbStruct

Input. Pointer to the db2DeactivateDbStruct structure.

**pSqlca** Output. Pointer to the sqlca structure.



## db2DeactivateDbStruct data structure parameters

### piDbAlias

Input. Pointer to the database alias name.

### piUserName

Input. Pointer to the user ID starting the database. Can be NULL.

### piPassword

Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

### iOptions

Input. Pre-defined value which is used explicitly to force the deactivation of the database and to denote that the deactivate command is being issued in the content of a specified list of members.

### iNumMembers

Input. Number of members listed in **iMemberList**.

### piMemberList

Input. Pointer to the member list.

## 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. Pointer to the sqlca structure.

## sqlg\_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.

## Examples

This section provides examples of different database deactivation scenarios. The following code block is a common body program that will be used for the database deactivation scenarios.

```
struct sqlca sqlca; // sqlca to carry the sqlcode
struct db2DeactivateDbStruct deactivateDbStruct;

strcpy(deactivateDbStruct->piUserName, "USER1"); //local instance
```

## sqle\_deactivate\_db - Deactivate database

The following deactivate API is shown common to the scenarios listed in the following examples:

```
db2DeactivateDb( db2Version, &deactivateDbStruct, &sqlca );
```

- Deactivate a database globally across all members

```
deactivateDbStruct.iOptions = DB2_ACTDEACT_DB_GLOBAL;  
deactivateDbStruct->piMemberList = NULL;
```

- Deactivate a database at a member

```
deactivateDbStruct.iOptions = DB2_ACTDEACT_DB_MEMBER;  
deactivateDbStruct->piMemberList->numMembers = 1;  
deactivateDbStruct->piMemberList->pMember[0] = 20;
```

- Deactivate a database at a specific member 30, using the FORCE option

```
deactivateDbStruct.iOptions = DB2_ACTDEACT_DB_MEMBER;  
deactivateDbStruct.iOptions |= DB2_DEACTIVATE_FORCE;  
deactivateDbStruct->piMemberList->numMembers = 1;  
deactivateDbStruct->piMemberList->pMember[0] = 30;
```

### 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 user application to disconnect will not shut the database down; **DEACTIVATE DATABASE** must be used in that case too.

### REXX API syntax

This API can be called from REXX through the SQLDB2 interface.

## sqleaddn - Add a database partition to the partitioned database environment

Adds a database partition to a database partition server.

### Scope

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

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqleaddn (
    void * pAddNodeOptions,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlgaddn (
    unsigned short addnOptionsLen,
    struct sqlca * pSqlca,
    void * pAddNodeOptions);
```

### sqleaddn API parameters

#### pAddNodeOptions

Input. A pointer to the optional `sqle_addn_options` structure. This structure is used to specify the source database partition server, if any, of the system temporary table space definitions for all database partitions to be created. 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

#### addnOptionsLen

Input. A 2-byte unsigned integer representing the length of the optional `sqle_addn_options` structure in bytes.

### Usage notes

This API should only be used if a database partition server is added to an environment that has one database and that database is not cataloged at the time of the add partition operation. In this situation, because the database is not cataloged, the add partition operation does not recognize the database, and does

## sqleaddn - Add a database partition to the partitioned database environment

not create a database partition for the database on the new database partition server. Any attempt to connect to the database partition on the new database partition server results in an error. The database must first be cataloged before the sqleaddn API can be used to create the database partition for the database on the new database partition server.

This API should not be used if the environment has more than one database and at least one of the databases is cataloged at the time of the add partition operation. In this situation, use the sqlecran API to create a database partition for each database that was not cataloged at the time of the add partition operation. Each uncataloged database must first be cataloged before the sqlecran API can be used to create the database partition for the database on the new database partition server.

Before adding a new database partition, ensure that there is sufficient storage for the containers that must be created.

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.

**Note:** Any uncataloged database is not recognized when adding a new database partition. The uncataloged database will not be present on the new database partition. An attempt to connect to the database on the new database partition returns the error message SQL1013N.

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 when the operation has completed.

The storage groups storage path definitions are retrieved when the sqleaddn API has to communicate with the catalog partition for each of the databases in the instance. 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.

## 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. The DB2 database system provides support for changing passwords on AIX, Linux and Windows operating systems.

### 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.

## sqleatcp - Attach to instance and change password

pSqlca

Output. A pointer to the sqlca structure.

### 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. In a DB2 pureScale environment, as indicated by **SQLWARN0** being set to 'W' and **SQLWARN4** being set to 'S', this value represents the member number. In a partitioned database environment, as indicated by token 10, this value represents the database partition number. If the database is neither in a partitioned environment nor in a DB2 pureScale environment, this value is always 0, if present.
9. In a DB2 pureScale environment, as indicated by **SQLWARN0** being set to 'W' and **SQLWARN4** being set to 'S', this value represents the number of members in a DB2 pureScale instance. If this value is non-zero, it represents the number of database partitions in a partitioned database environment. If the database is neither in a partitioned environment nor in a DB2 pureScale environment, this value is always 0. This token is present only with Version 5 or later.

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 previously).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

Certain functions (**db2start**, **db2stop**, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the **DB2INSTANCE** environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated, and where the password is changed, depend on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

## **REXX API syntax**

Calling this API directly from REXX is not supported. However, REXX programmers can use this function by calling the DB2 command line processor to execute the **ATTACH** command.

---

## 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. If the server instance operates in a DB2 pureScale environment, as indicated by SQLWARN0 being set to 'W' and SQLWARN4 being set to 'S', this value represents the member number. If, as indicated by token 9, the server instance operates in a partitioned environment, this token represents the node number. If the server instance operates in a non-partitioned environment and outside of a DB2 pureScale environment, this value is not applicable and is always 0.
9. If this value is zero, the server instance operates in a non-partitioned environment and outside of a DB2 pureScale environment. Otherwise, this non-zero value represents the number of members in a DB2 pureScale instance, if SQLWARN0 is set to 'W' and SQLWARN4 is to 'S'. If this value is non-zero but neither SQLWARN0 nor SQLWARN4 is set, it represents the number of nodes in a partitioned environment. This token is present only with Version 5 or later.

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 previously).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

## sqleatin - Attach to instance

Certain functions (**db2start**, **db2stop**, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the **DB2INSTANCE** environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated depends on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the `sqlesetc` API.

### 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.

## sqlecadb - 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 database partition server.

### 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 authorities:

- SYSADM
- SYSCTRL

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqlecadb (
    _SQLOLDCHAR * pDbName,
    _SQLOLDCHAR * pDbAlias,
    unsigned char Type,
    _SQLOLDCHAR * pNodeName,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    unsigned short Authentication,
    _SQLOLDCHAR * pPrincipal,
    struct sqlca * pSqlca);
```

```
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);
```

### sqlecadb API parameters

#### **pDbName**

Input. A string containing the database name.

#### **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.h`) 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 database partition 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 Linux and UNIX 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.h`) are:

#### **SQL\_AUTHENTICATION\_SERVER**

Specifies that authentication takes place on the database partition server containing the target database.

#### **SQL\_AUTHENTICATION\_CLIENT**

Specifies that authentication takes place on the database partition server where the application is invoked.

#### **SQL\_AUTHENTICATION\_KERBEROS**

Specifies that authentication takes place using Kerberos Security Mechanism.

## sqlcadb - Catalog a database in the system database directory

### SQL\_AUTHENTICATION\_NOT\_SPECIFIED

Authentication not specified.

### SQL\_AUTHENTICATION\_SVR\_ENCRYPT

Specifies that authentication takes place on the database partition server containing the target database, and that the authentication password is to be encrypted.

### SQL\_AUTHENTICATION\_DATAENC

Specifies that authentication takes place on the database partition server 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.

### SQL\_AUTHENTICATION\_SVRENC\_AES

Specifies that authentication takes place on the database partition server containing the target database, and that the authentication userid and password are to be encrypted using an Advanced Encryption Standard (AES) encryption algorithm.

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.

## sqlgcadb 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 0 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 0 if no path is provided.

### NodeNameLen

Input. A 2-byte unsigned integer representing the length in bytes of the node name. Set to 0 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.

## sqlecadb - Catalog a database in the system database directory

### 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

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 the shared cache (server only) in DB2, 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.

**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 database partition server containing the target database. This is the default.

**CLIENT** Authentication occurs at the database partition server 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 database partition server containing the target database, and that the authentication userid and password are to be encrypted.

**DATAENC**  
Specifies that authentication takes place on the database partition server 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.

**SQL\_AUTHENTICATION\_SVRENC\_AES0**  
Specifies that authentication takes place on the database partition server containing the target database, and that the authentication userid and password are to be encrypted using an AES encryption algorithm.

### 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.

## **sqlecadb - Catalog a database in the system database directory**

### **REXX examples**

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3  
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```



## sqlecran - 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 re-created. 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 re-create 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 authorities:

- 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
sqlecran (
    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);
```

### sqlecran 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.

## **sqlecran - 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.

## sqlecrea - Create database

Initializes a new database with an optional user-defined collating sequence, creates the three initial table spaces, creates the system tables, and allocates the recovery log.

### Scope

In a partitioned database environment, this API affects all database partition servers that are listed in the `db2nodes.cfg` file.

The database partition server from which this API is called becomes the catalog partition for the new database.

### Authorization

One of the following authorities:

- SYSADM
- SYSCTRL

**Note:** The effective user ID of the calling application or process is used for client-side authorization.

### 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
sqlecrea (
    char * pDbName,
    char * pLocalDbAlias,
    char * pPath,
    struct sqlbdbdesc * 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 sqlbdbdesc * pDbDescriptor,
    char * pPath,
    char * pLocalDbAlias,
    char * pDbName);
```

### sqlecrea 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 Linux and UNIX 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 sqldbterritoryinfo structure, containing the locale and the code set for the database. Can be NULL. The default code set for a database is UTF-8 (Unicode). If a particular code set and territory is needed for a database, the required code set and territory should be specified via the sqldbterritoryinfo structure. If this field is NULL, then one of the following options is allowed as a collation value for the database (sqlcode 1083): NULL, SQL\_CS\_SYSTEM, SQL\_CS\_IDENTITY\_16BIT, or SQL\_CS\_UNICODE.

**Important:** Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated in Version 10.1 and might be removed in a future release. For more information, see "Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated" at <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.wn.doc/doc/i0058749.html>.

**Reserved2**

Input. Reserved for future use.

**pDbDescriptorExt**

Input. This parameter refers to an extended database description block (sqldbdesext) 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 4096 bytes is chosen for the database and the default storage group, IBMSTOGROUP, is not created.

**pSqlca**

Output. A pointer to the sqlca structure.

**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

## sqlecrea - Create database

becomes the catalog partition for the new database. Two database partition groups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP.

- Binds the previously defined database manager bind files to the database (these are listed in db2ubind.lst). If one or more of these files do not bind successfully, sqlecrea returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC, if the **RESTRICTIVE** option is not selected.
- 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 authorities:
  - DBADM, CONNECT, CREATETAB, BINDADD, CREATE\_NOT\_FENCED, IMPLICIT\_SCHEMA, and LOAD authorities to the database creator
  - CONNECT, CREATETAB, BINDADD, and IMPLICIT\_SCHEMA authorities 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.

**Note:** If the **RESTRICTIVE** option is present, it causes the **restrict\_access** database configuration parameter to be set to YES and no privileges or authorities are automatically granted to PUBLIC. For more detailed information, see the **RESTRICTIVE** option of the **CREATE DATABASE** command.

With DBADM authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with SYSADM or DBAD, 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 xxxx 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.

The sqlecrea API accepts a data structure called the Database Descriptor Block (SQLEDBDESC). You can define your own collating sequence within this structure.

**Note:** You can only define your own collating sequence for a single-byte database.

To specify a collating sequence for a database:

- Pass the required SQLEDBDESC structure, or
- Pass a NULL pointer. The collating sequence of the operating system (based on the current locale code and the code page) is used. This is the same as specifying SQLDBCSS equal to SQL\_CS\_SYSTEM (0).

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).

#### **sqle850b**

If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

#### **sqle932a**

If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5035 (EBCDIC Japanese).

#### **sqle932b**

If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5026 (EBCDIC Japanese).

The collating sequence specified during database creation cannot be changed later. It determines how character strings are compared. This affects the structure of indexes as well as the results of queries. In a Unicode database, the catalog tables and views are always created with the `IDENTITY` collation, regardless of the collation specified in the create database call. In a non-Unicode database, the catalog tables and views are created with the database collation.

Use `sqlecadb` 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 Linux and UNIX operating 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 (table space 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)

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 usage, 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.

## **sqlecrea - Create database**

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.

## sqlectnd - 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 authorities:

- SYSADM
- SYSCTRL

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqlectnd (
    struct sqle_node_struct * pNodeInfo,
    void * pProtocolInfo,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlgctnd (
    struct sqlca * pSqlca,
    struct sqle_node_struct * pNodeInfo,
    void * pProtocolInfo);
```

### sqlectnd 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 Linux and UNIX operating systems, it is stored in the DB2 install directory (sql11ib, 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

## sqlectnd - Catalog an entry in the node 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 the shared cache (server only) in DB2, 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 TCPIP 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 TCPIP4 NODE nodename REMOTE hostname SERVER servicename  
[WITH comment]
```

### 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 TCPIP6 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.

### sqledcgd - 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 authorities:

- SYSADM
- SYSCTRL

#### Required connection

None

#### API include file

sqlenv.h

#### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqledcgd (
    _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);
```

#### sqledcgd 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.

## **sqlgdcgd 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.

---

## 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 authorities:

- 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.

---

## 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 authorities:

- SYSADM
- SYSCtrl

**Note:** The effective user ID of the calling application or process is used for client-side authorization.

### 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.

### 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 rollforward recovery after a restore operation, the files should be saved before 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.

## REXX API syntax

```
DROP DATABASE dbalias
```

## REXX API parameters

### dbalias

The alias of the database to be dropped.

---

## sqledrpn - 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 authorities:

- SYSADM
- SYSCtrl

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqledrpn (
    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);
```

### sqledrpn 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.

## **sqledrpn - 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 sqludrpt API or the ALTER DATABASE PARTITION GROUP statement.
3. Drop any event monitors that are defined on the database partition server.
4. Rerun sqledrpn 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.

---

### 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

## 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
sqlgfm (
    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.

---

# sqlefrce - 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 authorities:

- SYSADM
- SYSCTRL
- SYSMANT

## 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
sqlefrce (
    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);
```

## sqlefrce 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.



### 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 sqlefrce 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 example, if the user signs off between the time an agent ID is collected and sqlefrce is called. The user that calls this API is never forced off.

Agent IDs are recycled, and are used to force applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the wrong user may be forced.

## 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.

## **sqlefrce - Force users and applications off the system**

### **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 sqlefrce does not wait until all specified applications are terminated before returning.

## sqlgdad - 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 authorities:

- SYSADM
- SYSCTRL

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqlgdad (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlgddad (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
```

### sqlgdad 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 Power® System Servers

## sqlegdad - Catalog a database in the database connection services (DCS) directory

The database manager creates a Database Connection Services directory if one does not exist. This directory is stored on the path that contains the database manager instance that is being used. The DCS directory is maintained outside of the database.

The database must also be cataloged as a remote database in the system database directory.

**Note:** If directory caching is enabled, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications might not be effective until the application has restarted. To refresh the shared cache (server only) in DB2, 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.

## sqlegdcl - End a database connection services (DCS) directory scan

Frees the resources that are allocated by the sqlegdsc API.

### Authorization

None

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN  
sqlegdcl (  
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN  
sqlgdccl (  
    struct sqlca * pSqlca);
```

### sqlegdcl API parameters

pSqlca

Output. A pointer to the sqlca structure.

### REXX API syntax

CLOSE DCS DIRECTORY

## sqllegdel - 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 authorities:

- SYSADM
- SYSCTRL

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqllegdel (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggdel (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
```

### sqllegdel API parameters

#### pDCSDirEntry

Input/Output. A pointer to the Database Connection Services directory structure. Specify 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 sqleuncd API.

To recatalog a database in the DCS directory, use the sqllegdad API.

To list the DCS databases that are cataloged on a node, use the sqllegdsc, sqllegdgt, and sqllegdcl 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 the shared cache (server only) in DB2, stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another

## **sqlgdel - Uncatalog a database from the database connection services (DCS) directory**

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

## sqllegdgc - 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
sqllegdgc (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggdgc (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
```

### sqllegdgc 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



## **sqlegdgc - Get a specific entry in the database connection services (DCS) directory**

XXX.5 PARMS

XXX.6 COMMENT

XXX.7 RESERVED.

## sqlegdgt - 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
sqlegdgt (
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlgddgt (
    struct sqlca * pSqlca,
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries);
```

### sqlegdgt 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 sqlegdsc API, which returns the entry count, must be called before 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]
```

## **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

---

### sqlegdsc - 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 sqlegdgt API and sqlegdcl 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
sqlegdsc (
    short * pNumEntries,
    struct sqlca * pSqlca);
```

```
SQL_API_RC SQL_API_FN
sqlggsdsc (
    struct sqlca * pSqlca,
    short * pNumEntries);
```

#### sqlegdsc 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

## sqllegins - 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
sqllegins (
    _SQLOLDCHAR * pInstance,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlggins (
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pInstance);
```

### sqllegins 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 `sqleatin - 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.

---

## 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 sqleisig - Install Signal Handler, or a routine supplied by the programmer and installed using an appropriate operating system call.

### Authorization

None

### Required connection

None

### API include file

sqlenv.h

### 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 9. 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.

*Table 9. INTERRUPT actions (continued)*

Database activity	Action
CREATE DATABASE/CREATE DATABASE AT NODE/ADD NODE/DROP NODE VERIFY	After a certain point, these APIs are not interruptible. If the interrupt call is received before this point, the database is not created. If the interrupt call is received after this point, it is ignored.
DROP DATABASE/DROP DATABASE AT NODE	None. The APIs complete.
EXPORT/IMPORT/ RUNSTATS	Utility cancelled. Database updates rolled back.
FORCE APPLICATION	None. <b>FORCE APPLICATION</b> 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. <b>DROP DATABASE</b> 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 before invocation of the SQL statement.
Other SQL statements	Database transactions are set to the state existing before invocation of the SQL statement.

## REXX API syntax

INTERRUPT

### Examples

```
call SQLDBS 'INTERRUPT'
```

---

## sqleisig - Install signal handler

Installs the default interrupt (usually Control-C or Control-Break, or both) signal handler. When this default handler detects an interrupt signal, it resets the signal and calls sqleintr.

### Authorization

None

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqleisig (
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlgisig (
    struct sqlca * pSqlca);
```

### sqleisig 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 sqleintr API can be developed. Use either the operating system call or the language-specific library signal function. The sqleintr API should be the only database manager operation performed by a customized signal handler. Follow all operating system programming techniques and practices to ensure that the previously installed signal handlers work properly.

### REXX API syntax

INSTALL SIGNAL HANDLER



## sqlmgdb - Upgrade previous version of DB2 database to current version

Converts a previous (Version 8 or higher) version of a DB2 database to the current release.

The sqlmgdb and sqlgmdb APIs are deprecated and will be discontinued in a future release. You should use the new db2DatabaseUpgrade API instead.

### 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
sqlgmdb (
    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.

### sqlgmdb 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.

## sqlmgdb - Upgrade previous version of DB2 database to current version

### **UserNameLen**

Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero when no user name is supplied.

### **DbAliasLen**

Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

## **Usage notes**

This API will only upgrade a database to a newer version, and cannot be used to convert an upgraded 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 upgraded.

### **username**

User name under which the database is to be restarted.

### **password**

Password used to authenticate the user name.

---

## 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.

---

## 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 SLENINFO 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.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).

---

## 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.

---

## 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 Currentconnection setting for the CONNECTION type

XXX.2 Currentconnection setting for the SQLRULES

XXX.3 Currentconnection setting indicating which connections will be released when a COMMIT is issued.

XXX.4 Currentconnection setting of the **SYNCPPOINT** option. The **SYNCPPOINT** 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.6 Currentconnection setting for deferred PREPARE.

## 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.

---

## 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 (\_).

## **sqlesdeg - Set the maximum runtime intrapartition parallelism level or degree for SQL statements**

Sets the maximum run time degree of intrapartition 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 authorities:

- 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 intrapartition 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 example, 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.

## 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 required 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 according to:

- Whether type 2 CONNECTs are to be processed according to the DB2 rules or the Standard (STD) rules based on ISO/ANS SQL92.
- How an application specifies the format of LOB columns in the result set.

#### DB2

- Permits the SQL CONNECT statement to switch the current connection to another established (*dormant*) connection.
- This default setting allows an application to specify whether LOB values or LOB locators are retrieved only during the first fetch request. Subsequent fetch requests must use the same format for the LOB columns.

#### STD

- Permits the SQL CONNECT statement to establish a *new* connection only. The SQL SET CONNECTION statement must be used to switch to a dormant connection.
- The application can change between retrieving LOB values and LOB locators with each fetch request. This means that cursors with one or more LOB columns cannot be blocked, regardless of the **BLOCKING** bind option setting.

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.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

## sqleseti - Set client information

Permits an application to set client information (by setting the fields in the `sqle_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 before 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
sqleseti (
    unsigned short DbAliasLen,
    char * pDbAlias,
    unsigned short NumItems,
    struct sqle_client_info* pClient_Info,
    struct sqlca * pSqlca);
```

### sqleseti API parameters

#### **DbAliasLen**

Input. A 2-byte unsigned integer representing the length in bytes of the database alias. If a value greater than zero is provided, **pDbAlias** must point to the alias name, and the settings will affect only the specified connection. If zero is specified, the settings will affect all existing and future connections.

#### **pDbAlias**

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

**NumItems**

Input. Number of entries being modified. The minimum value is 1.

**pClient\_Info**

Input. A pointer to an array of **NumItems** `sqle_client_info` structures, each containing a type field indicating which value to set, the length of that value, and a pointer to the new value.

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

**Usage notes**

If an alias name was provided, a connection to the alias must already exist, and all connections to that alias will inherit the changes. The information will be retained until the connection for that alias is broken. If an alias name was not provided, settings for all existing connections will be changed, and any future connections will inherit the changes. The information will be retained until the program terminates.

The field names represent guidelines for the type of information that can be provided. For example, a TP monitor application could choose to provide the TP monitor transaction ID along with the application name in the `SQL_CLIENT_INFO_APPLNAM` field. This would provide better monitoring and accounting on the DB2 server, where the DB2 transaction ID can be associated with the TP monitor transaction ID.

Currently this API will pass information to DB2 OS/390 Version 5 and higher, DB2 Universal Database Version 7 and higher, and DB2 i5/OS™ V6R1 and higher. All information (except the accounting string) is displayed on the `DISPLAY THREAD` command, and will all be logged into the accounting records.

The data values provided with the API can also be accessed by SQL special register. The values in these registers are stored in the database code page. Data values provided with this API are converted to the database code page before being stored in the special registers. Any data value that exceeds the maximum supported size after conversion to the database code page will be truncated before being stored at the server. These truncated values will be returned by the special registers. The original data values will also be stored at the server and are not converted to the database code page. The unconverted values can be returned by calling the `sqleqryi` API.

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.

If the accounting string is set by this API, when changes are made to the client user ID or application name, the accounting string does not get updated. However, if the accounting string was set by the `wlm_set_client_info` procedure and the client user ID or application name is changed, then the accounting string will be updated.

---

## sqleuncd - Uncatalog a database from the system database directory

Deletes an entry from the system database directory.

### Authorization

One of the following authorities:

- SYSADM
- SYSCtrl

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqleuncd (
    _SQLOLDCHAR * pDbAlias,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlguncd (
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pDbAlias);
```

### sqleuncd 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 sqledrpd API.

To recatalog the database, use the sqlecadb 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 an earlier 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 the shared cache (server only) in DB2, 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.

## sqleuncn - Uncatalog an entry from the node directory

Deletes an entry from the node directory.

### Authorization

One of the following authorities:

- SYSADM
- SYSCtrl

### Required connection

None

### API include file

sqlenv.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqleuncn (
    _SQLLDCHAR * pNodeName,
    struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
sqlguncn (
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    _SQLLDCHAR * pNodeName);
```

### sqleuncn 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 sqlectnd 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 the shared cache (server only) in DB2, 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 NODE nodename

## **REXX API parameters**

### **nodename**

Name of the node to be uncataloged.

### 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.



---

## 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 required 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 required data.

### 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.

## 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.

If a positive line width is specified, LF/NULL sequences are inserted between words so that the lines do not exceed the line width.

## sqlogstt - Get the SQLSTATE message

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.

## sqludrdt - 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 database partitions to be moved based on the current data distribution. This API does not support the **NOT ROLLFORWARD RECOVERABLE** option of the **REDISTRIBUTE DATABASE PARTITION GROUP** command.

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 authorities:

- SYSADM
- SYSCTRL
- DBADM

In addition, one of the following groups of authorizations is also required:

- DELETE, INSERT, and SELECT privileges on all tables in the database partition group being redistributed
- DATAACCESS authority

### API include file

sqlutil.h

### API and data structure syntax

```
SQL_API_RC SQL_API_FN
sqludrdt (
    char * pNodeGroupName,
    char * pTargetPMapFileName,
    char * pDataDistFileName,
    SQL_PDB_NODE_TYPE * pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE * pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca * pSqlca);

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,
```

## sqludrdt - Redistribute data across a database partition group

```
SQL_PDB_NODE_TYPE * pDropList,  
unsigned short DropCount,  
unsigned char DataRedistOption,  
struct sqlca * pSqlca);
```

### sqludrdt API parameters

#### pNodeGroupName

The name of the database partition group to be redistributed.

#### 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,

## sqludrdt - Redistribute data across a database partition group

**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 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/sqlllib/redist directory on Linux and UNIX operating systems, using the following format for subdirectories and file name: *database-name.nodegroup-name.timestamp*.
- The \$HOME/sqlllib/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.

## **sqludrdt - Redistribute data across a database partition group**

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.



## sqlugrpn - Get the database partition server number for a row

Beginning with Version 9.7, this API is deprecated. Use the db2GetRowPartNum (Get the database partition server number for a row) API to return the database partition number and database partition server number for a row.

If you call the sqlugrpn API and the **DB2\_PMAP\_COMPATIBILITY** registry variable is set to OFF, the error message SQL2768N is returned.

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 code page 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,
```

## sqlugrpn - Get the database partition server number for a row

```
unsigned short territory_code,  
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);
```

### 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\_ctypecode

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.

## sqlugrpn - Get the database partition server number for a row

### SQL\_IMPLIEDDECIMAL\_FORMAT

The location of an implied decimal point is determined by the column definition. For example, if the column definition is DECIMAL(8,2), the value 12345 is processed as 123.45.

### SQL\_PACKEDDECIMAL\_FORMAT

All decimal column 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 corresponding 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.

---

## 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.

Beginning with DB2 9.7, this API is deprecated. Use the db2GetDistMap (Get distribution map) API to return the distribution information. If you call the sqlugtpi API and the **DB2\_PMAP\_COMPATIBILITY** registry variable is set to OFF, the error message SQL2768N is returned.

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 authorities:

- DATAACCESS 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.

---

## 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 authorities:

- 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.

## sqluvqdp - Quiesce table spaces for a table

This may be a two-part name with the schema and the table name separated by a period (.). If the schema is not provided, the CURRENT SCHEMA will be used.

The table cannot be a system catalog table. This field is mandatory.

### QuiesceMode

Input. Specifies the quiesce mode. Valid values (defined in sqlutil) are:

#### SQLU\_QUIESCEMODE\_SHARE

For share mode

#### SQLU\_QUIESCEMODE\_INTENT\_UPDATE

For intent to update mode

#### SQLU\_QUIESCEMODE\_EXCLUSIVE

For exclusive mode

#### SQLU\_QUIESCEMODE\_RESET

To reset the state of the table spaces to normal if either of the following conditions 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 in DB2 pureScale environments.

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.

## sqluvqdp - Quiesce table spaces for a table

When the quiesce exclusive request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer), however, has exclusive access to the table and the table spaces.

When a quiesce update request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces with the quiescer is recorded in the table space table.

There is a limit of five quiescers on a table space at any given time. Since QUIESCED EXCLUSIVE is incompatible with any other state, and QUIESCED UPDATE is incompatible with another QUIESCED UPDATE, the five quiescer limit, if reached, must have at least four QUIESCED SHARE and at most one QUIESCED UPDATE.

A quiescer can modify 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.



---

## Chapter 6. Calling DB2 APIs in REXX

Syntax for 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 can 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.

**Note:** For information about the supported versions of the REXX programming language, see: “Support for database application development in REXX” in *Getting Started with Database Application Development*.

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 differences:

- 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.
- 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<sup>(1)</sup> <sup>(2)</sup>
- Bind for DB2 Version 2 or 5<sup>(1)</sup>
- Create Database at Node
- Drop Database at Node
- Drop Node Verify
- Deactivate Database
- Deregister
- Load<sup>(3)</sup>

## Calling DB2 APIs in REXX

- Load Query
- Precompile Program<sup>(1)</sup>
- Rebind Package<sup>(1)</sup>
- 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, poLoadInfoOut 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 previously, 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.

---

## 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

None

### REXX API syntax

```
CHANGE SQLISL TO {RR|CS|UR|RS|NC}
```

### REXX API parameters

- |           |  |
|-----------|--|
| <b>RR</b> | Repeatable read.                       |
| <b>CS</b> | Cursor stability. This is the default. |
| <b>UR</b> | Uncommitted read.                      |
| <b>RS</b> | Read stability.                        |
| <b>NC</b> | No commit.                             |

## Change Isolation Level

---

## Chapter 7. Indoubt transaction management APIs

There might be times where it is useful to be able to query, commit, and roll back indoubt transactions without waiting for the Transaction Manager (TM) to perform a *re-sync* action. This situation might arise, for example, if the communication line is broken, and an indoubt transaction is tying up needed resources.

A set of APIs is provided for tool writers to perform heuristic functions on indoubt transactions when the resource owner (such as the database administrator) cannot wait for the TM to perform the *re-sync* operation. For the database manager, these resources include locks on tables and indexes, log space, and storage used by the transaction. Each indoubt transaction also decreases by one the maximum number of concurrent transactions that could be processed by the database manager.

The heuristic APIs have the capability to query, commit, and roll back indoubt transactions, and to cancel transactions that have been heuristically committed or rolled back, by removing the log records and releasing log pages.

**Attention:** The heuristic APIs should be used with caution and only as a last resort. The TM should drive the re-sync events. If the TM has an operator command to start the re-sync action, it should be used. If the user cannot wait for a TM-initiated re-sync, heuristic actions are necessary.

Although there is no set way to perform these actions, the following guidelines may be helpful:

- Use the `db2XaListIndTrans` function to display the indoubt transactions. They have a status = 'P' (prepared), and are not connected. The *gtrid* portion of an *xid* is the global transaction ID that is identical to that in other resource managers (RM) that participate in the global transaction.
- Use knowledge of the application and the operating environment to identify the other participating RMs.
- If the transaction manager is CICS®, and the only RM is a CICS resource, perform a heuristic rollback.
- If the transaction manager is not CICS, use it to determine the status of the transaction that has the same *gtrid* as does the indoubt transaction.
- If at least one RM has committed or rolled back, perform a heuristic commit or a rollback.
- If they are all in the prepared state, perform a heuristic rollback.
- If at least one RM is not available, perform a heuristic rollback.

If the transaction manager is available, and the indoubt transaction is due to the RM not being available in the second phase, or in an earlier re-sync, the DBA should determine from the TM's log what action has been taken against the other RMs, and then do the same. The *gtrid* is the matching key between the TM and the RMs.

Do not execute `sqlxhfrg` unless a heuristically committed or rolled back transaction happens to cause a log full condition. The `forget` function releases the log space occupied by this indoubt transaction. If a transaction manager eventually performs a re-sync action for this indoubt transaction, the TM could make the wrong decision to commit or to roll back other RMs, because no record was found in this

## **Indoubt transaction management APIs**

RM. In general, a missing record implies that the RM has rolled back.

## 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.

---

## 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

#### 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.

## db2XaListIndTrans - List indoubt transactions

**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 reactivated, committed, or rolled back at a later time. It is also possible that the transaction manager encountered an error and the transaction will not be completed. If this is the case, this transaction requires heuristic actions, because it may be holding locks and preventing other applications from accessing data.

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 roll back 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.

**pxid** 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 roll back 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.

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

## db2XaListIndTrans - List indoubt transactions

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 **oNumIndoubtsTotal**. If **oNumIndoubtsTotal** is greater than **oNumIndoubtsReturned**, the application can repeat these steps.

---

## 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.

---

## 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.

---

## 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.





---

## **Chapter 8. Threaded applications with concurrent access**

---

## sqliteAttachToCtx - 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  
sqliteAttachToCtx (  
    void * pCtx,  
    void * reserved,  
    struct sqlca * pSqlca);
```

### sqliteAttachToCtx API parameters

**pCtx** Input. A valid context previously allocated by `sqliteBeginCtx`.

**reserved**

Reserved for future use. Must be set to NULL.

**pSqlca**

Output. A pointer to the `sqlca` structure.

## sqlBeginCtx - 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 `sqlAttachToCtx` 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
sqlBeginCtx (
    void ** ppCtx,
    sqlint32 lOptions,
    void * reserved,
    struct sqlca * pSqlca);
```

### sqlBeginCtx 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 `sqlAttachToCtx` 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.

---

### 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.

## sqliteEndCtx - 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
sqliteEndCtx (
    void ** ppCtx,
    sqlint32 lOptions,
    void * reserved,
    struct sqlca * pSqlca);
```

### sqliteEndCtx API parameters

**ppCtx** Output. A data area in private memory (used for the storage of context information) that is freed.

#### lOptions

Input. Valid values are:

##### SQL\_CTX\_FREE\_ONLY

The context memory will be freed only if a prior detach has been done.

**Note:** **pCtx** must be a valid context previously allocated by `sqliteBeginCtx`.

##### SQL\_CTX\_END\_ALL

If necessary, a call to `sqliteDetachFromCtx` 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 `sqliteBeginCtx`. A call to `sqliteGetCurrentCtx` 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.

## sqleEndCtx - 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 `sqleEndCtx`.

---

## 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.

## 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.



## 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 1Options);
```

### sqlSetTypeCtx API parameters

#### 1Options

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.

**sqleSetTypeCtx - Set application context type**

---

## Chapter 9. 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 these 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 systems and a DLL on Windows 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.

---

### Enabling plug-ins

#### 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.

#### Before you begin

After you acquire a group retrieval plug-in that is suitable for your database management system, you can deploy it.

## Deploying a group retrieval plug-in

### 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.

## 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.

### Before you begin

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.

**Note:** You must stop the DB2 server or any applications using the plug-ins before you deploy a *new* version of an *existing* plug-in. Undefined behavior including traps will occur if a process is still using a plug-in when a new version (with the same name) is copied over it. This restriction is not in effect when you deploy a plugin for the first time or when the plug-in is not in use. After you acquire user ID/password authentication plug-ins that are suitable for your database management system, you can deploy them.

### 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.
- 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.

## 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.

### Before you begin

In the case of 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.

**Note:** You must stop the DB2 server or any applications using the plug-ins before you deploy a *new* version of an *existing* plug-in. Undefined behavior including traps will occur if a process is still using a plug-in when a new version (with the same name) is copied over it. This restriction is not in effect when you deploy a plug-in for the first time or when the plug-in is not in use.

## Deploying a GSS-API plug-in

After you acquire GSS-API authentication plug-ins that are suitable for your database management system, you can deploy them.

### 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 **srvcon\_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 **srvcon\_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 **srvcon\_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.
  2. 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:
  1. Copy the GSS-API authentication plug-in library in the client plug-in directory on the client, server, or gateway.
  2. Update the database manager configuration parameter **local\_gssplugin** with the name of the plug-in.
  3. Set the **authentication** database manager configuration parameter to GSSPLUGIN, or GSS\_SERVER\_ENCRYPT.

## Deploying a Kerberos plug-in

To customize the Kerberos authentication behavior of the DB2 security system, you can develop your own Kerberos authentication plug-ins or purchase one from a third party.

### Before you begin

If you want to deploy a new version of an existing plug-in, you must stop the DB2 server and any applications using the plug-in. Undefined behaviors, including traps, occur if a process is using a plug-in when you deploy a new version of that plug-in (with the same name).

## About this task

The Kerberos authentication plug-in can be deployed on a database server or a database client.

## 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 into the server plug-in directory.
  2. Update the setting of the **srvcon\_gssplugin\_list** database manager configuration parameter, which is 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 there is no Kerberos plug-in in the list, an error is returned. If there is more than one Kerberos plug-in in the list, an error is returned. If the configuration parameter value is blank and the **authentication** configuration parameter is set to KERBEROS or KRB\_SVR\_ENCRYPT, the default DB2 Kerberos plug-in, IBMkrb5, is used.
  3. If necessary, set the value of the **srvcon\_auth** database manager configuration parameter. If you want to deploy a Kerberos plug-in, the acceptable values for the **srvcon\_auth** database manager configuration parameter are as follows:
    - KERBEROS
    - KRB\_SERVER\_ENCRYPT
    - GSSPLUGIN
    - GSS\_SERVER\_ENCRYPT
    - Blank, but only if the **authentication** configuration parameter is set to one of the previous values in this list.
- To deploy a Kerberos authentication plug-in on a database client, perform the following steps on the client:
  1. Copy the Kerberos authentication plug-in library into the client plug-in directory.
  2. Set the **clnt\_krb\_plugin** database manager configuration parameter to the name of the Kerberos plug-in. If the value of the **clnt\_krb\_plugin** configuration parameter is blank, the client cannot use Kerberos authentication. On Windows, the default value is IBMkrb5. It only needs to be altered for a customized Kerberos plugin. On UNIX, the value must be set since the default value is blank. 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. Set the **clnt\_krb\_plugin** database manager configuration parameter to 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 use only a Kerberos authentication plug-in. The following example catalogs the testdb database:
 

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
          TARGET PRINCIPAL service/host@REALM
```

## Writing security plug-ins

### How DB2 loads security plug-ins

So that the DB2 database system has the necessary information to call security plug-in functions, a security plug-in must have a correctly set up initialization function.

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 plug-in 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 files
- 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
- Specify the pointers to the other functions in the library
- Specify 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.

### Restrictions for developing security plug-in libraries

There are certain restrictions that affect how you develop plug-in libraries.

The following list outlines the 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 the DB2 signal handlers. Interfering with the DB2 signal handlers could seriously interfere with the ability for DB2 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 the error handling used in DB2.

#### 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 dependent libraries are accessible through other methods, such as the following situations:

- 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, 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.

### 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 about 32-bit and 64-bit considerations 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
```

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 previously.

### 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 -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* 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.

**Fork** Plug-in libraries should not fork because file descriptors and sockets will be duplicated in the child process, and this can cause hangs or incorrect behavior. In particular, it can cause false file lock conflicts if child was

## Restrictions for developing security plug-in libraries

forked when we had an open file descriptor on that file. There is also the possibility that the fork will inherit many other resources like semaphores.

## Restrictions on security plug-ins

There are certain 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.

**Restrictions on the AUTHID identifier.** Version 9.5, and later, of the DB2 database system allows you to have an 128-byte authorization ID, but when the authorization ID is interpreted as an operating system user ID or group name, the operating system naming restrictions apply (for example, a limitation to 8 or 30 character user IDs and 30 character group names). Therefore, while you can grant an 128-byte authorization ID, it is not possible to connect as a user that has that authorization ID. If you write your own security plugin, you should be able to take full advantage of the extended sizes for the authorization ID. For example, you can give your security plug-in a 30-byte user ID and it can return an 128-byte authorization ID during authentication that you are able to connect with.

### InfoSphere® 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:

```
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.

## 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 10. 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_UNKNOWNERROR	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

Table 10. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-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
-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

## Return codes for security plug-ins

Table 10. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-23	DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME	GSS API plug-in only: the principal name is invalid.	db2secProcessServer PrincipalName
-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
-29	DB2SEC_PLUGIN_ROOT_NEEDED	The plug-in is trying to run an application that requires root privileges.	All
-30	DB2SEC_PLUGIN_UNEXPECTED_SYSTEM_ERROR	The plug-in encountered an unexpected system error. A possibility exists that the current system configuration is not supported.	All



## 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 example, 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 the **db2diag** log files. 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.

### Calling sequences for the security plug-in APIs

The sequence with which the DB2 database manager calls the security plug-in APIs varies according to the scenario in which the security plug-in API is called.

These are the main scenarios in which the DB2 database manager calls 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 calls 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 calls 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 negotiate user ID/password authentication (for example, when the **srvcon\_auth** parameter at the server is set to SERVER, SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP), the client application calls 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 negotiate userid/password authentication (for example, when the **srvcon\_auth** parameter at the server is set to SERVER; SERVER\_ENCRYPT, DATA\_ENCRYPT, or DATA\_ENCRYPT\_CMP), the client application calls the following security plug-in APIs:

- db2secRemapUserId();

### GSSAPI and Kerberos - implicit

On an implicit authentication, when the client and server negotiate GSS-API or Kerberos authentication (for example, when the **srvcon\_auth** parameter at the server is set to KERBEROS; KRB\_SERVER\_ENCRYPT, GSSPLUGIN, or GSS\_SERVER\_ENCRYPT), the client application calls the following security plug-in APIs. (The call to gss\_init\_sec\_context() uses 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 calls 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() might 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 calls the following security plug-in APIs:

- db2secGetDefaultLoginContext();
- db2secGetGroupsForUser();
- db2secFreeToken();
- db2secFreeGroupList();

## Calling sequences for the security plug-in APIs

These APIs are 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 calls 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 calls 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();
- db2secFreeGroupListMemory();

### 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 calls 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 calls the following security plug-in API with only the authorization ID as input:

- db2secGetGroupsForUser();

There will be no token from other security plug-ins.

---

## Security plug-ins

Authentication for the DB2 database system is done using *security plug-ins*. A security plug-in is a dynamically loadable library that provides authentication security services.

### Group retrieval plug-in

Retrieves group membership information for a particular user.

### User ID/password authentication plug-in

The following authentication types are implemented using a user ID and password authentication plug-in:

- CLIENT
- SERVER
- SERVER\_ENCRYPT

- DATA\_ENCRYPT
- DATA\_ENCRYPT\_CMP

These authentication types determine how and where authentication of a user occurs. The authentication type that is used is determined by the following method:

- For connect or attach operations, if you specify a value for the **srvcon\_auth** configuration parameter, then that value takes precedence over the value of the **authentication** configuration parameter.
- In all other cases, the value of the **authentication** configuration parameter is used.

### GSS-API authentication plug-in

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). The Kerberos protocol is the predominant means of implementing the GSS-API authentication mechanism. 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. Client-side Kerberos support is available on Solaris, AIX, HP-UX (64-bit only), Windows, and Linux operating systems. For Windows operating systems, Kerberos support is enabled by default.

The DB2 database manager supports these plug-ins at both the client and the server.

**Note:** Authentication types determine how and where a user is authenticated. To use a particular authentication type, set the value of the **authentication** database manager configuration parameter.

You can use each of the plug-ins independently, or with the other plug-ins. For example, you might use a specific sever-side authentication plug-in, but accept the DB2 default values for client and group authentication. Alternatively, you might have only a group retrieval, or a client authentication plug-in, but without a server-side plug-in.

If you want to use GSS-API authentication, plug-ins are required on both the client and the server.

The default behavior for authentication is to use a user ID/password plug-in that implements an operating-system-level mechanism to authenticate.

The DB2 database product includes plug-ins for group retrieval, user ID/password authentication, and GSS-API authentication. You can customize DB2 client and server authentication behavior further by developing your own plug-ins, or by purchasing plug-ins from a third party.

### Deployment of security plug-ins on DB2 clients

DB2 clients can support one group retrieval plug-in and one user ID/password authentication plug-in.

Alternatively, clients using GSS-API authentication plug-in determine which plug-in to use by scanning the list of implemented GSS-API plug-ins on the DB2 server. The first authentication plug-in name that matches a GSS-API authentication plug-in implemented on the client is the one chosen. You specify the list of implemented server GSS-API plug-ins using the **srvcon\_gssplugin\_list** database manager configuration parameter. The following figure portrays the security plug-in infrastructure on a DB2 client:

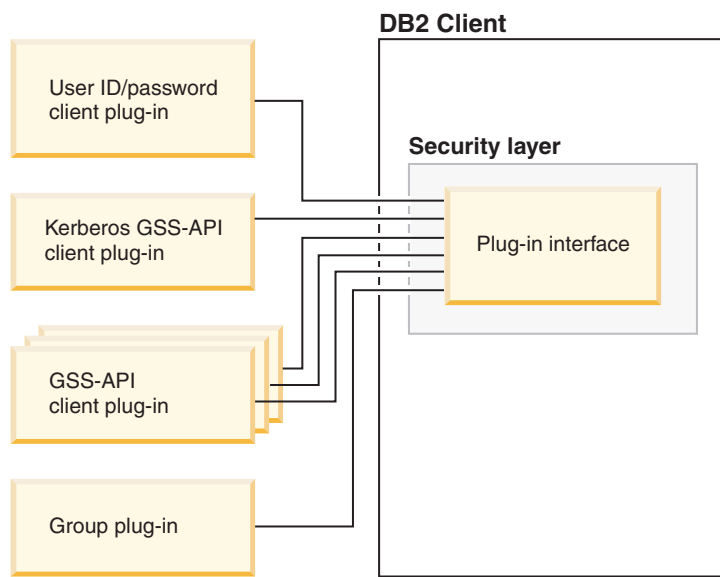


Figure 1. Deploying Security Plug-ins on DB2 Clients

### Deployment of security plug-ins on DB2 servers

DB2 servers can support one group retrieval plug-in, one user ID/password authentication plug-in, and multiple GSS-API plug-ins. You can specify the available GSS-API plug-ins as a list of values for the **srvcon\_gssplugin\_list** database manager configuration parameter. However, only one GSS-API plug-in in this list can be a Kerberos plug-in.

In addition to deploying the server-side security plug-ins on your database server, you might have to deploy client authentication plug-ins on your database server. When you run instance-level operations, such as the **db2start** and **db2trc** commands, the DB2 database manager performs authorization checking for these operations using client authentication plug-ins. Therefore, you might need to install the client authentication plug-in that corresponds to the authentication plug-in on the server. This plug-in name is specified by the **authentication** database manager configuration parameter on the server.

You can set the **authentication** and **srvcon\_auth** configuration parameters to different values. This scenario causes one mechanism to be used to authenticate database connections and the other mechanism to be used for local authorization.

The most common method for this approach is to:

- Set the **srvcon\_auth** configuration parameter to GSSPLUGIN; and
- Set the **authentication** configuration parameter to SERVER.

The **srvcon\_auth** configuration parameter is a means to override the authentication type used by incoming connections. These connections use the authentication method specified by the **srvcon\_auth** configuration parameter, but if this value is left empty, the value of the **authentication** parameter is used instead.

If you do not use client authentication plug-ins on the database server, instance-level operations, such as the **db2start** command, fail.

The following figure outlines the security authentication plug-in infrastructure on a DB2 server:

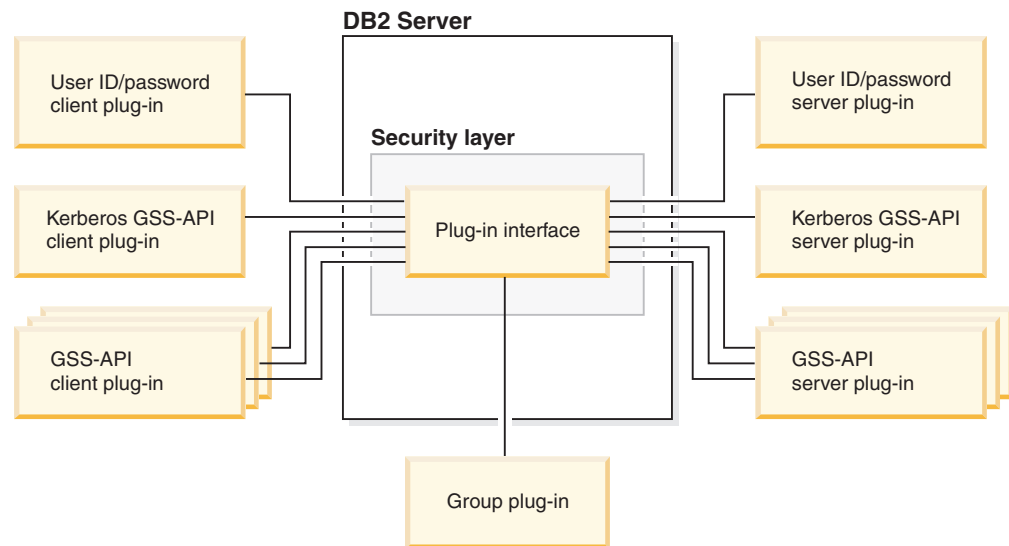


Figure 2. Deploying Security Plug-ins on DB2 Servers

## Enabling security plug-ins

You can specify the plug-ins to use for each authentication mechanism by setting database manager configuration parameters. The following table outlines these parameters:

Table 11. Database Manager configuration parameters for security authentication plug-ins

Description	Parameter name
Client Userid-Password Plugin	CLNT_PW_PLUGIN
Client Kerberos Plugin	CLNT_KRB_PLUGIN
Group Plugin	GROUP_PLUGIN
GSS Plugin for Local Authorization	LOCAL_GSSPLUGIN
Server Plugin Mode	SRV_PLUGIN_MODE
Server List of GSS Plugins	SRVCON_GSSPLUGIN_LIST
Server Userid-Password Plugin	SRVCON_PW_PLUGIN
Server Connection Authentication	SRVCON_AUTH
Database manager authentication	AUTHENTICATION

If you do not set the values for these parameters, the default plug-ins that the DB2 product supplies are used for group retrieval, user ID/password management, and Kerberos authentication (if the **authentication** parameter is set to KERBEROS on the server). However, a default GSS-API plug-in is not provided. Therefore, if you specify an authentication type of GSSPLUGIN for the **authentication** parameter, you must also specify a GSS-API authentication plug-in for the **srvcon\_gssplugin\_list** configuration parameter.

### Loading security plug-ins

All of the supported plug-ins that are identified by the database manager configuration parameters are loaded when the database manager starts.

During connect or attach operations, the DB2 client loads a plug-in that is appropriate for the security mechanism that the client negotiated with the server. 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. In this scenario, the client program makes an initial connection to server A that uses a GSS-API plug-in (G1). Server A sends a list of supported plug-ins to the client, and the matching G1 plug-in is loaded on the client. The client program then has another thread, which connects to server B that uses a GSS-API plug-in (G2). The client is informed about G2, which is then loaded, and now both G1 and G2 plug-ins are simultaneously in use on the client.

Actions other than connect or attach operations (such as updating the database manager configuration, starting and stopping the database manager, or turning a DB2 trace on and off) also require an authorization mechanism. For such actions, the DB2 client program loads a plug-in that is specified by another database manager configuration parameter:

- If you set the **authentication** configuration parameter to GSSPLUGIN, the DB2 database manager uses the plug-in specified by the **local\_gssplugin** configuration parameter.
- If you set the **authentication** configuration parameter to KERBEROS, the DB2 database manager uses the plug-in specified by the **clnt\_krb\_plugin** configuration parameter.
- Otherwise, the DB2 database manager uses the plug-in specified by the **clnt\_pw\_plugin** configuration parameter.

Security plug-ins are supported for connections made to the database server over both IPv4 and IPv6 address protocols.

### Developing security plug-ins

If you are developing a security authentication plug-in, you must implement the standard authentication functions used by the DB2 database manager. The functionality that you must implement for the three types of plug-ins:

#### Group retrieval plug-in

- Find and return the list of groups to which a user belongs

#### User ID/password authentication plug-in

- Identify the default security context (for a client plug-in only)
- Validate and, optionally, change a password



- Determine whether a particular string represents a valid user (for a server plug-in only)
- Modify the user ID or password that is provided on the client before it is sent to the server (for a client plug-in only)
- Return the DB2 authorization ID that is associated with a particular user

#### GSS-API authentication plug-in

- Identify the default security context (for a client plug-in only)
- Implement the required GSS-API functions
- Generate initial credentials based on a user ID and password and, optionally, change a password (for a client plug-in only)
- Create and accept security tickets
- Return the DB2 authorization ID that is associated with a particular GSS-API security context

You can pass a user ID of up to 255 characters for a connect statement that you issue through the CLP or via a dynamic SQL statement.

**Important:** The integrity of your DB2 database system installation can be compromised if security plug-ins are not adequately coded, reviewed, and tested. The DB2 database product takes precautions against many common types of failures, but it cannot guarantee complete integrity if user-written security plug-ins are deployed.

## Security plug-in library locations

After you acquire your security plug-ins (either 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

## Security plug-in library locations

**Note:** On Windows-based platforms, the subdirectories *instance name* and *group* are not created automatically. The instance owner has to manually create them.

## 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: .so

**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`

**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_SRVCN_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.

## Security plug-in support for two-part user IDs

The DB2 database manager on 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.

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 12. 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
Kerberos	IBMkrb5	IBMkrb5TwoPart

## Security plug-in support for two-part user IDs

**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.

## Security plug-in API versioning

The DB2 database system supports version numbering of the security plug-in 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 is the highest version number of the API that DB2 can support, and corresponds 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 specify the 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.

## 32-bit and 64-bit considerations for security plug-ins

In general, a 32-bit DB2 instance uses the 32-bit security plug-in and a 64-bit DB2 instance uses the 64-bit security plug-in. However, on a 64-bit instance, DB2 supports 32-bit applications, which 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 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 upgrade 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.

## 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 logs are a good resource for debugging and administrating security plug-ins. To see the an administration notification log file on UNIX, check `sqlllib/db2dump/instance name.N.nfy`. To see an 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.

## Security plug-in problem determination

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 about 32-bit and 64-bit considerations 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.

---

## Security plug-in APIs

To enable you to customize the DB2 database system authentication and group membership lookup behavior, the DB2 database system provides APIs that you can use to modify existing plug-in modules or build new security plug-in modules.

When you develop a security plug-in module, you need to implement the standard authentication or group membership lookup functions that the DB2 database manager will invoke. For the three available types of plug-in modules, 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 list shows 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.

**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 enter in the value for the security plug-in API parameter.

**Output**

Indicates that the security plug-in API will specify the value for the API parameter.



### APIs for group retrieval plug-ins

For the group retrieval plug-in module, you 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 the **db2diag** log files for debugging or informational purposes. This API is provided by the DB2 database system, 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,
        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    **errmsg,
db2int32 *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrorMsg)
(
char *msgtobefree
);

SQL_API_RC (SQL_API_FN * db2secPluginTerm)
(
char    **errmsg,
db2int32 *errmsglen
);

} 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.

### db2secDoesGroupExist API - Check if group exists

Determines whether 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 for Linux, UNIX, and Windows 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 *errormsglen );
```

#### 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.

##### errormsglen

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

### db2secFreeErrorMsg API - 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

##### **errorMsg**

Input. A pointer to the error message allocated from a previous API call.

### db2secFreeGroupListMemory API - 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 *errmsglen );
```

#### 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.

**errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in the **errmsg** parameter.

## db2secGetGroupsForUser API - 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 usernamespacelen,
  db2int32 usernamespacestype,
  const char *dbname,
  db2int32 dbname,
  void *token,
  db2int32 tokentype,
  db2int32 location,
  const char *authpluginname,
  db2int32 authpluginname,
  void **group,
  db2int32 *numgroups,
  char **errmsg,
  db2int32 *errmsglen );
```

### db2secGetGroupsForUser API parameters

**authid** Input. This parameter value is an SQL authid, which means that DB2 for Linux, UNIX, and Windows converts it to an uppercase character string with no trailing blanks. DB2 for Linux, UNIX, and Windows always provides 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 for Linux, UNIX, and Windows 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, see the Usage notes section in this topic.

#### authidlen

Input. Length in bytes of the **authid** parameter value. The DB2 database manager always provides 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 for Linux, UNIX, and Windows.

#### 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 the DB2 database manager.

## db2secGetGroupsForUser API - Get list of groups for user

### **usernamespacelen**

Input. Length in bytes of the **usernamespace** parameter value.

### **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, the DB2 database system 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 for Linux, UNIX, and Windows. 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 for Linux, UNIX, and Windows 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 for Linux, UNIX, and Windows if the **authid** is not authenticated (for example, if the **authid** does not match the current connected user).

**authpluginamelen**

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 containing concatenated varchars (a varchar is a character array in which the first byte indicates the number of bytes following the first byte). 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 for Linux, UNIX, and Windows 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.

**errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

### Usage notes

The following list describes the scenarios that which problems can occur if an incomplete group list is returned by this API to DB2 for Linux, UNIX, and Windows:

- 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.
- 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: DBADM, or CREATEIN (implicit or explicit on the jar schema). The operation will fail if the rights stated previously are granted to group containing the session authid, but not explicitly to the session authid.
  - Remove jar file. The session authid needs to have one of the following rights: DBADM, or DROPIN (implicit or explicit on the jar schema), or is the definer of the jar file. The operation will fail if the rights stated previously 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.
  - Replace jar file. This is same as removing the jar file, followed by installing the jar file. Both of the scenarios described previously apply.
- 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.

### db2secGroupPluginInit API - Initialize group plug-in

Initialization API, for the group-retrieval plug-in, that the DB2 database manager 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 the DB2 database manager 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 for Linux, UNIX, and Windows 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 the DB2 database system. The db2secGroupPluginInit API can call the db2secLogMessage API to log messages to the **db2diag** log files 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 files and the last two parameters 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 shows up in the db2diag log files 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 shows up in the **db2diag** log files only if the **diaglevel** database manager configuration parameter is set to 4.



## db2secGroupPluginInit API - Initialize group plug-in

### **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 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.

### db2secPluginTerm - Clean up group plug-in resources

Frees resources used by the group-retrieval plug-in.

This API is called by the DB2 database manager 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 example, 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 operating systems.

#### 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.

## APIs for user ID/password authentication plug-ins

For the user ID/password plug-in module, you 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 the **db2diag** log files for debugging or informational purposes. This API is provided by the DB2 database system, so you do not need to 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;
    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      *userspacelen,
        db2int32      *userspacetype,
        const char *dbname,
        db2int32      dbnamelen,
        void **token,
        char **errorMsg,
        db2int32      *errormsglen
    );
    /* Optional */
    SQL_API_RC (SQL_API_FN * db2secRemapUserid)
    (
        char          userid[DB2SEC_MAX_USERID_LENGTH],
        db2int32      *useridlen,
        char          usernamespace[DB2SEC_MAX_USERSPACE_LENGTH],
        db2int32      *userspacelen,
        db2int32      *userspacetype,
```

## APIs for user ID/password authentication plug-ins

```
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    usernamespace_type,
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
);

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_USERNAMESPACE_LENGTH],
db2int32    *usernamespacelen,
db2int32    *usernamespace_type,
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   usernamespace,
db2int32   usernamespace,
db2int32   usernamespace,
const char  *password,
db2int32   passwordlen,
const char  *newpassword,
db2int32   newpasswordlen,
const char  *dbname,
db2int32   dbname,
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)
(
char *errmsg
);

SQL_API_RC (SQL_API_FN * db2secFreeInitInfo)
(
void      *initInfo,
char      **errmsg,
db2int32  *errmsglen
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char      **errmsg,
db2int32  *errmsglen
);

/* 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>);

```

## APIs for user ID/password authentication plug-ins

```
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 a 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 the **db2diag** log files 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 the DB2 database system, so you do not need to implement them. The `db2secGetConDetails` API in turn, takes as its second parameter, **pConDetails**, a pointer to one of the following structures:

`db2sec_con_details_1:`

```
typedef struct db2sec_con_details_1
{
    db2int32 clientProtocol;
    db2UInt32 clientIPAddress;
    db2UInt32 connect_info_bitmap;
    db2int32 dbnameLen;
    char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

`db2sec_con_details_2:`

```
typedef struct db2sec_con_details_2
{
    db2int32 clientProtocol; /* See SQL_PROTOCOL_ in sqlenv.h */
    db2UInt32 clientIPAddress; /* Set if protocol is TCP/IP4 */
    db2UInt32 connect_info_bitmap;
    db2int32 dbnameLen;
    char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
    db2UInt32 clientIP6Address[4]; /* Set if protocol is TCP/IP6 */
} db2sec_con_details_2;
```

`db2sec_con_details_3:`

```
typedef struct db2sec_con_details_3
{
    db2int32 clientProtocol; /* See SQL_PROTOCOL_ in sqlenv.h */
    db2UInt32 clientIPAddress; /* Set if protocol is TCP/IP4 */
    db2UInt32 connect_info_bitmap;
```

```
db2int32 dbnameLen;
char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
db2UInt32 clientIP6Address[4]; /* Set if protocol is TCP/IP6 */
db2UInt32 clientPlatform; /* SQLM_PLATFORM_* from sqlmon.h */
db2UInt32 _reserved[16];
} db2sec_con_details_3;
```

The possible values for **conDetailsVersion** are `DB2SEC_CON_DETAILS_VERSION_1`, `DB2SEC_CON_DETAILS_VERSION_2`, and `DB2SEC_CON_DETAILS_VERSION_3` representing the version of the API.

**Note:** While using `db2sec_con_details_1`, `db2sec_con_details_2`, or `db2sec_con_details_3`, 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 Version 8.2 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 either the `db2sec_con_details_2` structure and the `DB2SEC_CON_DETAILS_VERSION_2` value, or the `db2sec_con_details_3` structure and the `DB2SEC_CON_DETAILS_VERSION_3` value.
- New plugins should use the `db2sec_con_details_3` structure and the `DB2SEC_CON_DETAILS_VERSION_3` 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_3` 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_3` structure. The **clientProtocol** field of the connection details structure will be set to one of `SQL_PROTOCOL_TCP/IP` (IPv4, with v1 of the structure), `SQL_PROTOCOL_TCP/IP4` (IPv4, with v2 of the structure) or `SQL_PROTOCOL_TCP/IP6` (IPv6, with v2 or v3 of the structure).
- The structure `db2sec_con_details_3` is identical to the structure `db2sec_con_details_2` except that it contains an additional field (**clientPlatform**) that identifies the client platform type (as reported by the communication layer) using platform type constants defined in `sqlmon.h`, such as `SQLM_PLATFORM_AIX`.

- `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 db2sec_userid_password_server_auth_functions_1
{
    db2int32 version;
    db2int32 plugin_type;

    /* parameter lists left blank for readability
       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>);
```

## APIs for user ID/password authentication plug-ins

```
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 a user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiServerAuthFunctions_1` structure.



## db2secClientAuthPluginInit API - Initialize client authentication plug-in

Initialization API, for the client authentication plug-in, that the DB2 database manager 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 the DB2 database manager currently supports. The DB2SEC\_API\_VERSION value (in db2secPlugin.h) contains the latest version number of the API that DB2 for Linux, UNIX, and Windows currently supports.

#### client\_fns

Output. A pointer to memory provided by the DB2 database manager 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 contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/password authentication plug-in. In future versions of DB2 for Linux, UNIX, and Windows, 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 the DB2 database manager 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 the DB2 database manager. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to the **db2diag** log files for debugging or informational purposes. The first parameter (**level**) of db2secLogMessage

## db2secClientAuthPluginInit API - Initialize client authentication plug-in

API specifies the type of diagnostic errors that will be recorded in the **db2diag** log files and the last two parameters 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 **db2diag** log files only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the **diaglevel** database manager configuration parameter. For example, if you use the DB2SEC\_LOG\_INFO value, the message text will appear in the **db2diag** log files only 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.

## **db2secClientAuthPluginTerm API - Clean up client authentication plug-in resources**

Frees resources used by the client authentication plug-in.

This API is called by the DB2 database manager 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 example, 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 operating systems.

### **API and data structure syntax**

```
SQL_API_RC ( SQL_API_FN *db2secClientAuthPluginTerm)
( char      **errmsg,
  db2int32 *errmsglen);
```

### **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.

#### **errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

### db2secDoesAuthIDExist - Check if authentication ID exists

Determines if the **authid** represents an individual user (for example, 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,
  dbint32 authidlen,
  char      **errmsg,
  dbint32 *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.

## **db2secFreeInitInfo API - 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 the DB2 database manager. 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.

### db2secFreeToken API - Free memory held by token

Frees the memory held by a token. This API is called by the DB2 database manager 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.

## db2secGenerateInitialCred API - Generate initial credentials

The db2secGenerateInitialCred API obtains the initial GSS-API credentials based on the user ID and password that are passed in.

For Kerberos, this is 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, the DB2 database manager specifies 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 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 *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.

#### usernamespacelen

Input. Length in bytes of the **usernamespace** parameter value.

#### usernamespacetype

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 honor 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.

## db2secGenerateInitialCred API - Generate initial credentials

### **newpasswordlen**

Input. Length in bytes of the **newpassword** parameter value.

### **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.

### **dbname1en**

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 for Linux, UNIX, and Windows. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. The DB2 database manager calls 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 be returned only if there is an internal error in the API that prevented it from completing properly.

### **errmsg1en**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.



## db2secGetAuthIDs API - 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 dbname1en,
  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 *username1en,
  db2int32 *initsessionidtype,
  char **errormsg,
  db2int32 *errormsglen );
```

### db2secGetAuthIDs API parameters

#### userid

Input. The authenticated user. This is usually not used for GSS-API authentication unless a trusted context is defined to permit switch user operations without authentication. In those situations, the user name provided for the switch user request is passed in this parameter.

#### useridlen

Input. Length in bytes of the **userid** parameter value.

#### usernamespace

Input. The namespace from which the user ID was obtained.

#### usernamespace1en

Input. Length in bytes of the **usernamespace** parameter value.

#### usernamespacectype

Input. Namespace type value. Currently, 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.

#### dbname1en

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

## db2secGetAuthIDs API - Get authentication IDs

db2secValidatePassword API is not called. In environments where a trusted context is defined that allows switch user operations without authentication, the db2secGetAuthIDs API must be able to accommodate receiving a NULL value for this token parameter and be able to derive a system authorization ID based on the **userid** and **useridlen** input parameters mentioned previously.

### **SystemAuthID**

Output. The system authorization ID that corresponds to the ID of the authenticated user. The size is 255 bytes, but the DB2 database manager 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 example, when issuing a SET SESSION AUTHORIZATION statement. The size is 255 bytes, but the DB2 database manager 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 be used only for auditing and will be logged in the "User ID" field in the audit record for CONNECT statement. If the API does specify the **username** parameter, the DB2 database manager copies it from the **userid**.

### **usernameLen**

Output. Length in bytes of the **username** parameter value.

### **initSessionIDtype**

Output. Session authid type indicating whether 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)

### **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.

## db2secGetDefaultLoginContext API - Get default login context

Determines the user associated with the default login context, that is, 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_USERNAMESPACE_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 user ID. In particular, a plug-in that does not use the UNIX or Linux identity of the user to establish the DB2 authorization ID 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

## db2secGetDefaultLoginContext API - Get default login context

limitation that the **usernamespace** parameter must be set to the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

### **usernamespace**

Output. Namespace type value. Currently, 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.

### **dbname**

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.

### **errmsg**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

## db2secProcessServerPrincipalName API - Process service principal name returned from server

The db2secProcessServerPrincipalName API 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 the **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.

### db2secRemapUserId API - Remap user ID and password

This API is called by the DB2 database manager on the client side to remap a given user ID and password (and possibly new password and username space) to values different from those given at connect time.

The DB2 database manager only calls 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 is not 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 usernamepace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32 *usernamepacelen,
  db2int32 *usernamepacetype,
  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.

##### usernamepace

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 **usernamepace** will be used by the DB2 database manager only for CLIENT type authentication and is disregarded for other authentication types.

##### usernamepacelen

Input or output. Length in bytes of the **usernamepace** parameter value. Under the limitation that the **usernamepacetype** parameter must be set to the value DB2SEC\_NAMESPACE\_SAM\_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

##### usernamepacetype

Input or output. Old and new namespace type value. Currently, 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

## db2secRemapUserid API - Remap user ID and password

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.

### **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 is passed by the DB2 database manager 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.

### **dbname1en**

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.

### **errmsg1en**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

### db2secServerAuthPluginInit - Initialize server authentication plug-in

The db2secServerAuthPluginInit API is the initialization API for the server authentication plug-in that the DB2 database manager 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
(
    dbint32 version,
    void *server_fns,
    db2secGetConDetails *getConDetails_fn,
    db2secLogMessage *logMessage_fn,
    char **errorMsg,
    dbint32 *errormsglen );
```

#### db2secServerAuthPluginInit API parameters

##### version

Input. The highest version number of the API that the DB2 database manager currently supports. The DB2SEC\_API\_VERSION value (in db2secPlugin.h) contains the latest version number of the API that the DB2 database manager currently supports.

##### server\_fns

Output. A pointer to memory provided by the DB2 database manager 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 contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/password authentication plug-in.

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 the DB2 database manager 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



## db2secServerAuthPluginInit - Initialize server authentication plug-in

**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.

### 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 the DB2 database manager. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to the **db2diag** log files 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**

## db2secServerAuthPluginInit - Initialize server authentication plug-in

log files and the last two parameters 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 appear in the **db2diag** log files 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 appear in the **db2diag** log files only 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.

### **errormsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

## **db2secServerAuthPluginTerm API - Clean up server authentication plug-in resources**

The db2secServerAuthPluginTerm API frees resources used by the server authentication plug-in.

This API is called by the DB2 database manager 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 example, 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 operating systems.

### **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.

### db2secValidatePassword API - 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 previous scenarios 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 usernamespacetype,
  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.

##### 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, the DB2 database system only supports the value `DB2SEC_NAMESPACE_SAM_COMPATIBLE`. When the user ID is not available, the **usernamespace** parameter is set to the value `DB2SEC_USER_NAMESPACE_UNDEFINED` (defined in `db2secPlugin.h`).

### **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`.

### **dbname1en**

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 rightmost 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 operating-system-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 the DB2 database manager 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 the DB2 database manager is calling from the server side or client side to validate

## db2secValidatePassword API - Validate password

password. If this bit value is set, then the DB2 database manager is calling from server side; otherwise, it is calling from the client side.

The DB2 database system default behavior for an implicit authentication is to allow the connection without any password validation. However, plug-in developers can 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.

### **errmsglen**

Output. A pointer to an integer that indicates the length in bytes of the error message string in **errmsg** parameter.

## Required APIs and definitions for GSS-API authentication plug-ins

The following table 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 13. Required APIs and Definitions for GSS-API authentication plug-ins

API type	API name	Description
Client-side APIs	<code>gss_init_sec_context</code>	Initiate a security context with a peer application.
Server-side APIs	<code>gss_accept_sec_context</code>	Accept a security context initiated by a peer application.
Server-side APIs	<code>gss_display_name</code>	Convert an internal format name to text.
Common APIs	<code>gss_delete_sec_context</code>	Delete an established security context.
Common APIs	<code>gss_display_status</code>	Obtain the text error message associated with a GSS-API status code.
Common APIs	<code>gss_release_buffer</code>	Delete a buffer.
Common APIs	<code>gss_release_cred</code>	Release local data structures associated with a GSS-API credential.
Common APIs	<code>gss_release_name</code>	Delete internal format name.
Required definitions	<code>GSS_C_DELEG_FLAG</code>	Requests delegation.
Required definitions	<code>GSS_C_EMPTY_BUFFER</code>	Signifies that the <code>gss_buffer_desc</code> does not contain any data.
Required definitions	<code>GSS_C_GSS_CODE</code>	Indicates a GSS major status code.
Required definitions	<code>GSS_C_INDEFINITE</code>	Indicates that the mechanism does not support context expiration.
Required definitions	<code>GSS_C_MECH_CODE</code>	Indicates a GSS minor status code.
Required definitions	<code>GSS_C_MUTUAL_FLAG</code>	Mutual authentication requested.
Required definitions	<code>GSS_C_NO_BUFFER</code>	Signifies that the <code>gss_buffer_t</code> variable does not point to a valid <code>gss_buffer_desc</code> structure.
Required definitions	<code>GSS_C_NO_CHANNEL_BINDINGS</code>	No communication channel bindings.
Required definitions	<code>GSS_C_NO_CONTEXT</code>	Signifies that the <code>gss_ctx_id_t</code> variable does not point to a valid context.
Required definitions	<code>GSS_C_NO_CREDENTIAL</code>	Signifies that <code>gss_cred_id_t</code> variable does not point to a valid credential handle.
Required definitions	<code>GSS_C_NO_NAME</code>	Signifies that the <code>gss_name_t</code> variable does not point to a valid internal name.
Required definitions	<code>GSS_C_NO_OID</code>	Use default authentication mechanism.

## Required APIs and definitions for GSS-API authentication plug-ins

Table 13. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

API type	API name	Description
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.



## Restrictions for GSS-API authentication plug-ins

The following list describes the 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. The DB2 database manager 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, the DB2 database manager 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, the DB2 database manager only sends a token if `gss_init_sec_context()` fails and generates a token on the first call.

## Security plug-in samples

UNIX and Linux directories: 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 14. Security plug-in sample program files*

Sample program name	Program description
<code>combined.c</code>	Combined user ID and password authentication and group lookup sample
<code>duaconfigschemaldif</code>	Used to ITDS LDAP server when a HP-UX system wants to connect to the LDAP server.
<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>IBMLDAPauthclient.c</code>	Implements a client side DB2 security plugin that interacts with an LDAP user registry
<code>IBMLDAPauthserver.c</code>	Implements a server side DB2 security plugin that interacts with an LDAP user registry
<code>IBMLDAPconfig.c</code>	Contains functions related to finding and parsing the configuration file for a DB2 LDAP security plugin
<code>IBMLDAPgroups.c</code>	Implements a DB2 security plugin for LDAP-based group lookup
<code>IBMLDAPutils.c</code>	Contains utility functions used in the DB2 LDAP security plugin
<code>IBMLDAPutils.h</code>	LDAP security plugin header file
<code>JCCKerberosPlugin.java</code>	Implements a GSS-API Plugin that does Kerberos authentication using IBM DB2 Universal Driver
<code>JCCKerberosPluginTest.java</code>	Uses JCCKerberosPlugin to get a DB2 Connection using IBM DB2 Universal Driver.
<code>JCCSimpleGSSPlugin.java</code>	Implements a GSS-API Plugin that does userid and password checking using IBM DB2 Universal Driver.
<code>JCCSimpleGSSContext.java</code>	Implements a GSSContext to be used by JCCSimpleGSSPlugin
<code>JCCSimpleGSSCredential.java</code>	Implements a GSSCredential to be used by JCCSimpleGSSPlugin
<code>JCCSimpleGSSException.java</code>	Implements a GSSException to be used by JCCSimpleGSSPlugin
<code>JCCSimpleGSSName.java</code>	Implements a GSSName to be used by JCCSimpleGSSPlugin
<code>JCCSimpleGSSPluginTest.java</code>	Uses JCCSimpleGSSPlugin to get a DB2 Connection using IBM DB2 Universal Driver.

---

## 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.

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 enter 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 enter in the value for the API parameter.

### 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.

**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.

## Return codes

The following table lists all possible return codes for this API.

Table 15. db2VendorGetNextObj API return codes

Number	Return code	Explanation
2	SQLUV_COMM_ERROR	Communication error with device - Failure.
4	SQLUV_INV_ACTION	Invalid action requested or combination of input parameters results in an operation that is not possible - Failure.
5	SQLUV_NO_DEV_AVAIL	No device is available for use at the moment - Failure.
6	SQLUV_OBJ_NOT_FOUND	No object found - Failure.
12	SQLUV_INV_DEV_HANDLE	Invalid device handle - Failure.
14	SQLUV_END_OF_DATA	No more query objects to return - Success.
18	SQLUV_DEV_ERROR	Device error - Failure.
19	SQLUV_WARNING	Warning, see return code - Success.
21	SQLUV_MORE_DATA	More query objects to return - Success.
25	SQLUV_IO_ERROR	I/O error - Failure.
30	SQLUV_UNEXPECTED_ERROR	A severe error encountered - Failure.

## **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.

## 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 16. 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

### 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 *hdl,
               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 )

**hdl** 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 the DB2 database system, 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 data set, 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 17. 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	

### 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 * hdlc,  
               struct Data *data,  
               struct Return_code *return_code);
```

#### sqluvget API parameters

**hdlc** 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 18. 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, <b>action</b> = SQLU_ABORT (see following note)	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, <b>action</b> = SQLU_ABORT (see following note)	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, <b>action</b> = SQLU_ABORT (see following note)	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, <b>action</b> = SQLU_ABORT (see following note)	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, <b>action</b> = SQLU_ABORT (see following note)	The session will be terminated.

Table 18. 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, <b>action</b> = SQLU_ABORT	
SQLUV_LINK_NOT_EXIST	No link currently exists	sqluvend, <b>action</b> = SQLU_ABORT (see following note)	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, <b>action</b> = SQLU_ABORT (see following note)	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.

## 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 following table).

The Init\_input structure contains elements that can be used by the vendor product to determine if the backup or restore can proceed:

#### size\_HI\_order and size\_LOW\_order

This is the estimated size of the backup. They can be used to determine if the vendor devices can handle the size of the backup image. They can be used to estimate the quantity of removable media that will be required to hold the backup. It might be beneficial to fail at the first sqluvint call if problems are anticipated.

#### req\_sessions

The number of user requested sessions can be used in conjunction with the estimated size and the prompting level to determine if the backup or restore operation is possible.

#### prompt\_lvl

The prompting level indicates to the vendor if it is possible to prompt for actions such as changing removable media (for example, put another tape

in the tape drive). This might suggest that the operation cannot proceed since there will be no way to prompt the user. If the prompting level is WITHOUT PROMPTING and the quantity of removable media is greater than the number of sessions requested, DB2 will not be able to complete the operation successfully.

DB2 names the backup being written or the restore to be read via fields in the DB2\_info structure. In the case of an **action** = SQLUV\_READ, the vendor product must check for the existence of the named object. If it cannot be found, the return code should be set to SQLUV\_OBJ\_NOT\_FOUND so that DB2 will take the appropriate action.

After initialization is completed successfully, DB2 will continue by calling other data transfer APIs, but may terminate the session at any time with an sqluvend call.

## Return codes

Table 19. 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 <b>action</b> = SQLUV_WRITE, the next call will be to the sqluvput API (to BACKUP data). If <b>action</b> = SQLUV_READ, verify the existence of the named object before 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.
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.

## sqluvint - Initialize and link to a vendor device

Table 19. Valid Return Codes for sqluvint and Resulting DB2 Action (continued)

Literal in Header File	Description	Probable Next Call	Other Comments
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 <b>action</b> 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 <b>action</b> 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.
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.

Table 19. 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 <b>action</b> = SQLUV_WRITE (BACKUP), the next call will be to sqluvput API. If <b>action</b> = SQLUV_READ, verify the existence of the named object before 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.

### 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 *hdlc,  
              struct Data *data,  
              struct Return_code *return_code);
```

#### sqluvput API parameters

**hdlc** 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 20. Valid Return Codes for sqluvput and Resulting DB2 Action

Literals 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, <b>action</b> = SQLU_ABORT (see following note).	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, <b>action</b> = SQLU_ABORT (see following note).	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, <b>action</b> = SQLU_ABORT (see following note).	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, <b>action</b> = SQLU_ABORT (see following note).	The session will be terminated.
SQLUV_ENDOFMEDIA	End of media reached, for example, end of tape.	sqluvend	



*Table 20. Valid Return Codes for sqluvput and Resulting DB2 Action (continued)*

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_DATA_RESEND	Device requested to have buffer sent again.	sqluvput	DB2 will retransmit the last buffer. This will only be done once.
SQLUV_DEV_ERROR	Device error.	sqluvend, <b>action</b> = SQLU_ABORT (see following note).	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, <b>action</b> = SQLU_ABORT (see following note).	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, <b>action</b> = SQLU_ABORT (see following note).	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.

## 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 21. 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 <b>server_id</b> , <b>db2instance</b> , <b>dbname</b> , and <b>timestamp</b> 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 21. 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 <b>action</b> is SQLUV_WRITE: 0 - full database backup 3 - table space level backup. When <b>action</b> 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 21. Fields in the DB2\_info Structure. (continued)

Field Name	Data Type	Description
dbPartition	db2NodeType	DB partition number. Numbers in the range {0 - 999} 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;
    db2NodeType dbPartition;
} DB2_info ;
```

## 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 22. 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

```
typedef struct Vendor_info
{
    char *vendor_id;
```

## Vendor\_info

```
char *version;  
char *release;  
char *level;  
char *server_id;  
sqlint32 max_bytes_per_txn;  
sqlint32 num_objects_in_backup;  
void *reserve;  
} Vendor_info;
```

## Init\_input

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 23. 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 <b>VendorOptionsSize</b> parameter.
size_HI_order	squint32	High order 32 bits of DB size estimate in bytes; total size is 64 bits.
size_LOW_order	squint32	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; this means that 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 <b>pVendorOptions</b> 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

```
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;
```



## 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 24. 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 ;
```

## Data

Contains data transferred between DB2 and a vendor device.

This structure is used by the `sqluvput` API when data is being written to the vendor device and by the `sqluvget` API when data is being read from the vendor device.

*Table 25. Fields in the Data Structure*

Field Name	Data Type	Description
obj_num	sqlint32	The sequence number assigned by DB2 during a backup operation.
buff_size	sqlint32	The size of the buffer.
actual_buff_size	sqlint32	The actual number of bytes sent or received. This must not exceed <b>buff_size</b> .
dataptr	void	Pointer to the data buffer. DB2 allocates space for the buffer.
reserve	void	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;
```

## 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 26. Fields in the Return\_code Structure*

Field Name	Data Type	Description
return_code (see following note)	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;
```

---

## 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
- 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 database products.

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

## DB2 APIs for using compression with backup and restore operations

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

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.

### 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;
```

## 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.



## 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 want 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.

## 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 log2 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.

## Compress - Compress a block of data

### **tgtSize**

Input. Size in bytes of the target buffer.

### **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.

## 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.

### 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.

### 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.

### InitCompression - Initialize the compression library

Initializes the compression library. DB2 will pass the db2Info and piInfo structures. The library will enter 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.



## 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.

### 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 COMPRESS_CB *pCB);
```

#### TermCompression API parameters

##### **pCB**

Input. This is the control block that was returned by the InitCompression API call.

## 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.



---

## Chapter 10. Data structures used by APIs

---

## db2DistMapStruct

This structure is used by the db2GetDistMap API

Table 27. Fields in the db2DistMapStruct structure

Field Name	Data Type	Description
tname	unsigned char	Fully qualified table name.
partinfo	db2PartitioningInfo	This structure is used to store partitioning information, such as the distribution map and the distribution key of a table.

### API and data structure syntax

```
SQL_STRUCTURE db2DistMapStruct
{
    unsigned char          *tname;    /* Fully qualified table    */
                                /* name                      */
    struct db2PartitioningInfo *partinfo; /* Partitioning            */
                                /* Information              */
};
```

## db2HistoryData

This structure is used to return information after a call to the db2HistoryGetEntry API.

**Note:** The db2HistoryGetEntry API is only supported in C, C++, or Java programming languages. It is no longer supported in COBOL, FORTRAN and REXX programming languages. You can issue a query to access database history records by using the DB\_HISTORY administrative view.

Table 28. Fields in the db2HistoryData structure

Field name	Data type	Description
<b>ioHistDataID</b>	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.
<b>oObjectPart</b>	db2Char	The first 14 characters are a time stamp with format <i>yyyymmddhhmmss</i> , 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.
<b>oEndTime</b>	db2Char	A time stamp with format <i>yyyymmddhhmmss</i> , indicating when the operation was completed.
<b>oID</b>	db2Char	Unique backup or table identifier.
<b>oTableQualifier</b>	db2Char	Table qualifier.
<b>oTableName</b>	db2Char	Table name.
<b>oLocation</b>	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 <b>oObjectPart</b> 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 <b>oLocation</b> is interpreted differently, depending on <b>oDeviceType</b> parameter: <ul style="list-style-type: none"> <li>• For disk or diskette (D or K), a fully qualified file name.</li> <li>• For tape (T), a volume label.</li> <li>• For TSM (A and F), the vendor library name/path that did the backup.</li> <li>• For user exit or other (U or O), free form text.</li> </ul>
<b>oComment</b>	db2Char	Free form text comment.
<b>oCommandText</b>	db2Char	Command text, or DDL.
<b>oLastLSN</b>	db2LSN	Last log sequence number.
<b>oEID</b>	Structure	Unique entry identifier.
<b>poEventSQLCA</b>	Structure	Result sqlca of the recorded event.
<b>poTablespace</b>	db2Char	A list of table space names.
<b>iNumTablespaces</b>	db2UInt32	Number of entries in the <b>poTablespace</b> list that are available for use by the db2HistoryGetEntry API.

## db2HistoryData

Table 28. Fields in the db2HistoryData structure (continued)

Field name	Data type	Description
<b>oNumTablespaces</b>	db2UInt32	Number of entries in the <b>poTablespace</b> 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.
<b>ioLogRange</b>	db2HistoryLogRange	Log ranges for each log stream.
<b>oOperation</b>	char	See Table 29.
<b>oObject</b>	char	Granularity of the operation: D for full database, P for table space, and T for table.
<b>oOptype</b>	char	See Table 30.
<b>oStatus</b>	char	Entry status: A for active; I for inactive; E for expired; D for deleted; and X for do not delete.
<b>oDeviceType</b>	char	Device type. This field determines how the <b>oLocation</b> field is interpreted: A for TSM, C for client, D for disk, F for snapshot backup, 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 29. 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
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

Table 30. Valid **oOptype** values in the db2HistData structure

<b>oOperation</b>	<b>oOptype</b>	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



Table 30. Valid **oOptype** values in the **db2HistData** structure (continued)

<b>oOperation</b>	<b>oOptype</b>	<b>Description</b>	<b>C/COBOL/FORTRAN definition</b>
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	offline, online, incremental offline, incremental online	DB2HISTORY_OPTYPE_OFFLINE, DB2HISTORY_OPTYPE_ONLINE, DB2HISTORY_OPTYPE_INCR_OFFLINE, DB2HISTORY_OPTYPE_INCR_ONLINE
T	C R	add containers, rebalance	DB2HISTORY_OPTYPE_ADD_CONT, DB2HISTORY_OPTYPE_REB

Table 31. Fields in the **db2HistoryEID** structure

<b>Field name</b>	<b>Data type</b>	<b>Description</b>
<b>ioNode ioHID</b>	SQL_PDB_NODE_TYPE db2UInt32	Node number. Local database history records entry ID.

This structure is used in the **db2HistoryData** structure

Table 32. Fields in the **db2HistoryLogRange** structure

<b>Field name</b>	<b>Data type</b>	<b>Description</b>
<b>iNumLogStreams</b>	DB2UINT32	Allocated number of log streams in <b>oStream</b> .
<b>oNumLogStreams</b>	DB2UINT32	Number of valid log streams being returned in <b>oStream</b> .
<b>oStream</b>	db2HistoryLogStreamRange	List of log ranges (by log stream).

This structure is used in the **db2HistoryLogRange** structure

Table 33. Fields in the **db2HistoryLogStreamRange** structure

<b>Field name</b>	<b>Data type</b>	<b>Description</b>
<b>oStreamID</b>	db2LogStreamIDType	Log stream ID of the log range

Table 33. Fields in the db2HistoryLogStreamRange structure (continued)

Field name	Data type	Description
<b>oFirstLog</b>	DB2UINT32	<p>The earliest log file extent.</p> <ul style="list-style-type: none"> <li>• Required to apply rollforward recovery for an online backup.</li> <li>• Required to apply rollforward recovery for an offline backup.</li> <li>• Applied after restoring a full database or table space level backup that was current when the load started.</li> </ul>
<b>oLastLog</b>	DB2UINT32	<p>The latest log file extent.</p> <ul style="list-style-type: none"> <li>• Required to apply rollforward recovery for an online backup.</li> <li>• Required to apply rollforward recovery to the current point in time for an offline backup.</li> <li>• Applied after restoring a full database or table space level backup that was current when the load operation finished.</li> </ul>

## API and data structure syntax

```
typedef SQL_STRUCTURE db2HistoryData
{
    char ioHistDataID[8];
    db2Char oObjectPart;
    db2Char oEndTime;
    db2Char oID;
    db2Char oTableQualifier;
    db2Char oTableName;
    db2Char oLocation;
    db2Char oComment;
    db2Char oCommandText;
    db2LSN oLastLSN;
    db2HistoryEID oEID;
    struct sqlca *poEventSQLCA;
    struct db2Char *poTablespace;
    db2UInt32 iNumTablespaces;
    db2UInt32 oNumTablespaces;
    db2HistoryLogRange ioLogRange;
    char oOperation;
    char oObject;
    char oOptype;
    char oStatus;
    char oDeviceType;
    db2HistoryData;
}
typedef SQL_STRUCTURE db2HistoryLogRange
{
    db2UInt32 iNumLogStreams;
    db2UInt32 oNumLogStreams;
    struct db2HistoryLogStreamRange *oStream;
```

```

    db2HistoryLogRange;
}

typedef SQL_STRUCTURE db2HistoryLogStreamRange
{
    db2LogStreamIDType oStreamID;
    db2UInt32 oFirstLog;
    db2UInt32 oLastLog;
    db2HistoryLogStreamRange;
}

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 database history records entry ID.

### db2HistoryLogRange data structure parameters

#### **iNumLogStreams**

Allocated number of db2HistoryLogStreamRange's in **oStream**

#### **oNumLogStreams**

Number of valid log streams being returned in **oStream**.

#### **oStream**

List of log ranges (by log stream).

---

## db2LSN data structure

This union, used by the db2ReadLog and db2ReadLogNoConn APIs, 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 34. Fields in the db2LSN Structure*

Field Name	Data Type	Description
lsnU64	db2UInt64	Specifies the log sequence number.

### API and data structure syntax

```
typedef SQL_STRUCTURE db2LSN
{
    db2UInt64    lsnU64;    /* Log sequence number        */
} db2LSN;
```

## sql\_dir\_entry

This structure is used by the DCS directory APIs.

Table 35. Fields in the SQL-DIR-ENTRY Structure

Field Name	Data Type	Description
STRUCT_ID RELEASE CODEPAGE COMMENT LDB TDB AR PARM	SMALLINT 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).
```

\*

## SQLB\_TBS\_STATS

This structure is used to return additional table space statistics to an application program.

Table 36. 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 <b>TOTALPAGES</b> minus (overhead plus partial extents). For SMS, equal to <b>TOTALPAGES</b> .
USEDPAGES	INTEGER	For DMS, the total number of pages in use. For SMS, equal to <b>TOTALPAGES</b> .
FREEPAGES	INTEGER	For DMS, equal to <b>USEABLEPAGES</b> minus <b>USEDPAGES</b> . For SMS, not applicable.
HIGHWATERMARK	INTEGER	For DMS, the high water mark is the current "end" of the table space address space. This refers to 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 usable 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 usable pages.

### API and data structure syntax

```
SQL_STRUCTURE SQLB_TBS_STATS
{
    sqluint32 totalPages;
    sqluint32 useablePages;
    sqluint32 usedPages;
    sqluint32 freePages;
    sqluint32 highWaterMark;
};
```

### COBOL Structure

```
* File: sqlutil.cbl
01 SQLB-TBS-STATS.
   05 SQL-TOTAL-PAGES          PIC 9(9) COMP-5.
   05 SQL-USEABLE-PAGES        PIC 9(9) COMP-5.
   05 SQL-USED-PAGES           PIC 9(9) COMP-5.
   05 SQL-FREE-PAGES           PIC 9(9) COMP-5.
   05 SQL-HIGH-WATER-MARK      PIC 9(9) COMP-5.
*
```

## SQLB\_TBSCONTQRY\_DATA

This structure is used to return container data to an application program.

Table 37. 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.
*
```



## SQLB\_TBSPQRY\_DATA

This structure is used to return table space data to an application program.

Table 38. 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	INTEGER (64-BIT)	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.

Table 38. 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 before a 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;
    sqluint64 lifeLSN;
    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 table space that the quiesced object was created in.

#### quiesceObject

Input. Object ID of the quiesced object.

### COBOL Structure

```
* File: sqlutbsp.cbl
01 SQLB-TBSPQRY-DATA.
   05 SQL-TBSPQVER          PIC X(8).
   05 SQL-ID                PIC 9(9) COMP-5.
   05 SQL-NAME-LEN          PIC 9(9) COMP-5.
   05 SQL-NAME              PIC X(128).
   05 SQL-TOTAL-PAGES       PIC 9(9) COMP-5.
   05 SQL-USEABLE-PAGES     PIC 9(9) COMP-5.
   05 SQL-FLAGS             PIC 9(9) COMP-5.
   05 SQL-PAGE-SIZE         PIC 9(9) COMP-5.
   05 SQL-EXT-SIZE          PIC 9(9) COMP-5.
   05 SQL-PREFETCH-SIZE     PIC 9(9) COMP-5.
   05 SQL-N-CONTAINERS      PIC 9(9) COMP-5.
   05 SQL-TBS-STATE         PIC 9(9) COMP-5.
   05 SQL-LIFE-LSN          PIC 9(18) COMP-5.
   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).
```

\*

## 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).
*
```

---

## sqlchar

This structure is used to pass variable length data to the database manager.

*Table 39. 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 output 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).
```

## 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    sqldaaid[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
{

```

## SQLDA

```
    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.
*
```



## sqldcol

This structure is used to pass variable column information to the db2Export, db2Import, and db2Load APIs.

Table 40. 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 <b>DCOLNAME</b> .
DCOLNAME	Array	An array of <b>DCOLNUM</b> sqldcol 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 `sqllocpair` structures. The number of elements in this array is determined by the **dcolnum** field of the `sqldcol` structure, which must be positive. Each element in the array is a pair of 2-byte integers that indicate where the column begins and ends. The first element of each location pair is the byte within the file where the

column begins, and the second element is the byte where the column ends. The first byte position within a row in the file is considered byte position 1. The columns can overlap.

### SQL\_METH\_D

Default. When importing or loading DEL and IXF files, the first column of the file is loaded or imported into the first column of the table, and so on. When 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 41. Fields in the SQLDCOLN Structure

Field Name	Data Type	Description
DCOLNLEN	SMALLINT	Length of the data pointed to by <b>DCOLNPTR</b> .
DCOLNPTR	Pointer	Pointer to a data element determined by <b>DCOLMETH</b> .

**Note:** The **DCOLNLEN** and **DCOLNPTR** fields are repeated for each column specified.

Table 42. Fields in the SQLLOCTAB Structure

Field Name	Data Type	Description
LOCPAIR	Array	An array of sqllocpair structures.

Table 43. 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;
};

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.
*
```

## sqle\_addn\_options

This structure is used to pass information to the sqleaddn API.

Table 44. 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 following section 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 <b>tblspace_type</b> 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;
};
```

## 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).
```

## 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 255 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 45. 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 <b>pValue</b> must be large enough to contain the maximum length for the type. A length of zero indicates a null value.
PVALUE	Pointer	Pointer to an application-allocated buffer that contains the specified value. The data type of this value is dependent on the type field.

The valid entries for the SQLE-CLIENT-INFO **TYPE** element and the associated descriptions for each entry are listed in the following table:

Table 46. Connection Settings

Type	Data Type	Description
SQL_CLIENT_INFO_USERID	CHAR(255)	The user ID for the client. Some servers may truncate the value. This user ID is for identification purposes only, and is not used for any authorization.
SQL_CLIENT_INFO_WRKSTNNAME	CHAR(255)	The workstation name for the client. Some servers may truncate the value.
SQL_CLIENT_INFO_APPLNAME	CHAR(255)	The application name for the client. Some servers may truncate the value.
SQL_CLIENT_INFO_PROGRAMID	CHAR(80)	The program identifier for the client. Once this element is set, DB2 Universal Database 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.
SQL_CLIENT_INFO_ACCTSTR	CHAR(255)	The accounting string for the client. Some servers may truncate the value.
SQL_CLIENT_INFO_AUTOCOMMIT	CHAR(1)	The autocommit setting of the client. It can be set to SQL_CLIENT_AUTOCOMMIT_ON or SQL_CLIENT_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 sql_client_info
{
    unsigned short    type;
    unsigned short    length;
    char *pValue;
};
```

### COBOL Structure

```
* File: sqlenv.cbl
01 SQL-CLIENT-INFO.
   05 SQL-CLIENT-INFO-ITEM OCCURS 4 TIMES.
      10 SQL-CLIENT-INFO-TYPE   PIC S9(4) COMP-5.
      10 SQL-CLIENT-INFO-LENGTH PIC S9(4) COMP-5.
      10 SQL-CLIENT-INFO-VALUE  USAGE IS POINTER.
*
```

## sqlc\_conn\_setting

This structure is used to specify connection setting types and values for the sqlcqryc and sqlcsetc APIs.

Table 47. 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 in the following table (defined in sqlenv and sql):

Table 48. 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.
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.



Table 48. Connection Settings (continued)

Type	Value	Description
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.
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 <b>DB2NODE</b> . 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 <b>DB2NODE</b> . 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.

## sqle\_conn\_setting

### API and data structure syntax

```
SQL_STRUCTURE sqle_conn_setting
{
    unsigned short    type;
    unsigned short    value;
};
```

### COBOL Structure

```
* File: sqleenv.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.
*
```

## sqle\_node\_local

This structure is used to catalog local nodes for the sqlectnd API.

Table 49. 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.
*
```

## sqle\_node\_npipe

This structure is used to catalog named pipe nodes for the sqlectnd API.

Table 50. 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.
*
```

## sqle\_node\_struct

This structure is used to catalog nodes for the sqlectnd API.

Table 51. Fields in the SQLE-NODE-STRUCT Structure

Field Name	Data Type	Description
<b>STRUCT_ID</b>	SMALLINT	Structure identifier.
<b>CODEPAGE</b>	SMALLINT	Code page for comment.
<b>COMMENT</b>	CHAR(30)	Optional description of the node.
<b>NODENAME</b>	CHAR(8)	Local name for the node where the database is located.
<b>PROTOCOL</b>	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\_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).
*
```

## sqle\_node\_tcpip

This structure is used to catalog TCP/IP nodes for the sqlectnd 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\_TCPIP, SQL\_PROTOCOL\_TCPIP4 or SQL\_PROTOCOL\_TCPIP6 in the SQLE-NODE-STRUCT structure before calling the sqlectnd 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 in the SQLE-NODE-STRUCT structure before calling the sqlectnd API. SOCKS is not supported on IPv6. For example, SQL\_PROTOCOL SOCKS with an IPv6 address is not supported.

Table 52. Fields in the SQLE-NODE-TCPIP Structure

Field Name	Data Type	Description
<b>HOSTNAME</b>	CHAR(255)	Hostname or IP address on which the DB2 server instance resides. The type of IP address accepted depends on the protocol selected.
<b>SERVICE_NAME</b>	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 sqle_node_tcpip
{
    _SQLOLDCHAR hostname[SQL_HOSTNAME_SZ+1];
    _SQLOLDCHAR service_name[SQL_SERVICE_NAME_SZ+1];
};
```

### COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-TCPIP.
   05 HOSTNAME          PIC X(255).
   05 FILLER            PIC X.
   05 SERVICE-NAME      PIC X(14).
   05 FILLER            PIC X.
*
```

## sqledbdesc

The Database Description Block (SQLEDBDESC) structure can be used during a call to the sqlecrea API to specify permanent values for database attributes. These attributes include database comment, collating sequences, and table space definitions.

Table 53. 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 <code>SQLC_DBDESC_2</code> (defined in <code>sqlenv</code> ). 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. Note: Specify <code>SQL_CS_NONE</code> to specify that the collating sequence for the database is <code>IDENTITY</code> (which implements a binary collating sequence). <code>SQL_CS_NONE</code> is the default.
SQLDBUDC	CHAR(256)	If <b>SQLDBCSS</b> is set to <code>SQL_CS_USER</code> , the <i>n</i> th byte of this field contains the sort weight of the code point whose underlying decimal representation is <i>n</i> in the code page of the database. If <b>SQLDBCSS</b> is set to <code>SQL_CS_UNICODE</code> , this field contains the language-aware or locale-sensitive UCA-based collation name (a NULL terminated string up to 128 bytes in length). If <b>SQLDBCSS</b> is not equal to <code>SQL_CS_USER</code> or <code>SQL_CS_UNICODE</code> , this field is ignored.
SQLDBCMT	CHAR(30)	The comment for the database.
SQLDBSGP	INTEGER	Reserved field. No longer used.

Table 53. 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: <b>SQLDBNSG</b> 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 <b>SQLTSEXT</b> and <b>SQLDBNSG</b> 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 <b>SQLTSEXT</b> and <b>SQLDBNSG</b> 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 <b>SQLTSEXT</b> and <b>SQLDBNSG</b> will be created.

The table space description block structure (SQLETSDESC) is used to specify the attributes of any of the three initial table spaces.



Table 54. 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_DBTSDDESC_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 <b>dft_extent_sz</b> 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 <b>dft_prefetch_sz</b> configuration parameter.
SQLFSCACHING	UNSIGNED CHAR	File system caching. If a value of 1 is supplied, file system caching will be OFF for the current table space. If a value of 0 is supplied, file system caching will be ON for the current table space. Specify 2 to indicate the default setting. In this case, file system caching will be OFF on AIX, Linux, Solaris, and Windows except on AIX JFS, Linux on System z®, Solaris non-VxFS for SMS temporary table space files, and for SMS Large Object files or Large Files. File system caching will be ON for all other platforms.
SQLPOVHD	DOUBLE	Table space I/O usage, 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.

Table 54. Fields in the *SQLETSDESC* Structure (continued)

Field Name	Data Type	Description
SQLTSTYP	CHAR(1)	Indicates whether the table space is system-managed or database-managed. .
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 <b>sqlccnt</b> <i>SQLETSDESC</i> structures.

Table 55. Fields in the *SQLETSDESC* Structure

Field Name	Data Type	Description
SQLCTYPE	CHAR(1)	Identifies the type of this container. .
SQLCSIZE	INTEGER	Size of the container identified in <b>SQLCONTR</b> , specified in 4KB pages. Valid only when <b>SQLTSTYP</b> is set to <b>SQL_TBS_TYP_DMS</b> .
SQLCLEN	SMALLINT	Length of following <b>SQLCONTR</b> value.
SQLCONTR	CHAR(256)	Container string.

Valid values for **SQLCBCSS** (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. If you pass a **NULL** pointer, the collating sequence of the operating system (based on the current locale code and the code page) is used. This is the same as specifying **SQLCBCSS** equal to **SQL\_CS\_SYSTEM** (0).

#### **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 website ([www.unicode.org](http://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 4.0.0 with normalization implicitly set to 'on'. Details of the UCA can be found in the Unicode Technical Standard #10 available at the Unicode Consortium website ([www.unicode.org](http://www.unicode.org)). This value can only be specified when creating a Unicode database.

**Important:** Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated in Version 10.1 and might be removed in a future release. For more information, see “Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated” at <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.wn.doc/doc/i0058749.html>.

#### SQL\_CS\_UCA400\_LSK

The UCA (Unicode Collation Algorithm) collation sequence that is based on the Unicode Standard version 4.0.0 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 website ([www.unicode.org](http://www.unicode.org)). This value can only be specified when creating a Unicode database.

**Important:** Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated in Version 10.1 and might be removed in a future release. For more information, see “Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated” at <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.wn.doc/doc/i0058749.html>.

#### SQL\_CS\_UCA400\_LTH

UCA (Unicode Collation Algorithm) collation sequence based on the Unicode Standard version 4.0.0, 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 website ([www.unicode.org](http://www.unicode.org)). This value can only be specified when creating a Unicode database.

**Important:** Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated in Version 10.1 and might be removed in a future release. For more information, see “Collations based on the Unicode Collation Algorithm of the Unicode Standard version 4.0.0 have been deprecated” at <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.wn.doc/doc/i0058749.html>.

#### SQL\_CS\_UNICODE

Collating sequence is language-based for a Unicode database. The specific collation name is specified in the **SQLDBUDC** field and must be terminated with a 0x00 byte. The collation name can identify any language-aware collation as defined in "Language-aware collations for Unicode data" or any locale-sensitive UCA-based collation identified in "Unicode Collation Algorithm based collations".

For example, to use collation equivalent to US English in code page 819, set **SQLDBCSS** to **SQL\_CS\_UNICODE** and **SQLDBUDC** to **SYSTEM\_819\_US**.

**Note:** When **CREATE DATABASE** is performed against a server earlier than Version 9.5, this option cannot be used. By default, a Unicode database on such a server will be created with **SYSTEM** collation.

Valid values for **SQLTSTYP** (defined in `sqlenv`) are:

**SQL\_TBS\_TYP\_SMS**

System managed

**SQL\_TBS\_TYP\_DMS**

Database managed

Valid values for **SQLCTYPE** (defined in `sqlenv`) are:

**SQL\_TBSC\_TYP\_DEV**

Device. Valid only when **SQLTSTYP** = **SQL\_TBS\_TYP\_DMS**.

**SQL\_TBSC\_TYP\_FILE**

File. Valid only when **SQLTSTYP** = **SQL\_TBS\_TYP\_DMS**.

**SQL\_TBSC\_TYP\_PATH**

Path (directory). Valid only when **SQLTSTYP** = **SQL\_TBS\_TYP\_SMS**.

## API and data structure syntax

```
SQL_STRUCTURE sqledbdesc
{
    _SQLOLDCHAR sqldbaid[8];
    sqlint32 sqldbaccp;
    sqlint32 sqldbcss;
    unsigned char sqldbudc[SQL_CS_SZ];
    _SQLOLDCHAR sqldbcmt[SQL_CMT_SZ+1];
    _SQLOLDCHAR pad[1];
    sqluint32 sqldbsgp;
    short sqldbnsq;
    char pad2[2];
    sqlint32 sqltsext;
    struct SLETSDESC *sqlcatts;
    struct SLETSDESC *sqlusrts;
    struct SLETSDESC *sqltmpts;
};

SQL_STRUCTURE SLETSDESC
{
    char sqltsdid[8];
    sqlint32 sqlextnt;
    sqlint32 sqlprftc;
    double sqlpovhd;
    double sqltrfrt;
    char sqltstyp;
    unsigned char sqlfscaching;
    short sqlccnt;
    struct SLETSDESC containr[1];
};
```

```
SQL_STRUCTURE SQLETSDESC
{
    char sqlctype;
    char pad1[3];
    sqlint32 sqlcsize;
    short sqlclen;
    char sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
    char pad2[2];
};
```

### **sqledbdesc 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.

### **SQLETSDESC 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 SQLTSEXT                PIC S9(9) COMP-5.
    05 SQLCATTS                USAGE IS POINTER.
    05 SQLUSRTS                USAGE IS POINTER.
    05 SQLTMPTS                USAGE IS POINTER.
```

\*

```
* File: sqletsd.cbl
01 SQLETSDESC.
    05 SQLTSDID                PIC X(8).
    05 SQLEXTNT                PIC S9(9) COMP-5.
    05 SQLPRFTC                PIC S9(9) COMP-5.
    05 SQLPOVHD                USAGE COMP-2.
    05 SQLTRFRT                USAGE COMP-2.
    05 SQLTSTYP                PIC X.
    05 SQL-PAD1                PIC X.
    05 SQLCCNT                PIC S9(4) COMP-5.
    05 SQL-CONTAINR OCCURS 001 TIMES.
        10 SQLCTYPE            PIC X.
        10 SQL-PAD1            PIC X(3).
        10 SQLCSIZE            PIC S9(9) COMP-5.
        10 SQLCLEN             PIC S9(4) COMP-5.
        10 SQLCONTR            PIC X(256).
        10 SQL-PAD2            PIC X(2).
```

\*

## sqledbdesc

```
* File: sqlenv.cbl
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).
*
```

## sqledbdescext

The extended database description block (sqledbdescext) structure is used during a call to the sqlecrea API to specify permanent values for database attributes.

The extended database description block creates the default storage group, 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 (sqledbdesc) structure.

If this structure is not passed to the sqlecrea API, the following behavior is used:

- The default storage group, IBMSTOGROUP, is created.
- 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

**Note:** Although, you can create a database specifying the AUTOMATIC STORAGE NO clause, the AUTOMATIC STORAGE clause is deprecated and might be removed from a future release.

### API and data structure syntax

```
SQL_STRUCTURE sqledbdescext
{
    sqluint32 sqlPageSize;
    struct sqlAutoStorageCfg *sqlAutoStorage;
    struct SQLETSDESCEXT *sqlcattsext;
    struct SQLETSDESCEXT *sqlusrsext;
    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;
};
```

**sqledbdescext data structure parameters***Table 56. Fields in the sqledbdescext 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, the default storage group is created and a single storage path is assumed with a value determined by the dbpath passed in, or the database manager configuration parameter, <b>dftdbpath</b> .
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).
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 sqlebdbdescext structure, and specifies the storage paths for the default storage group.

Table 57. 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 <b>SQLSTORAGEPATHS</b> array. If the value is 0, the <b>SQLSTORAGEPATHS</b> pointer must be NULL. The maximum number of storage paths is 128 (SQL_MAX_STORAGE_PATHS).
SQLSTORAGEPATHS	Pointer	An array of string pointers that point to storage paths. The number of pointers in the array is reflected by <b>SQLNUMSTORAGEPATHS</b> . Set <b>SQLSTORAGEPATHS</b> to NULL if there are no storage paths being provided (in which case, <b>SQLNUMSTORAGEPATHS</b> must be set to 0). The maximum length of each path is 175 characters.

Valid values for **SQLENABLEAUTOSTORAGE** (defined in sqleenv) are:

### SQL\_AUTOMATIC\_STORAGE\_NO

The default storage group is not created and table spaces managed by automatic storage cannot be created. When this value is used, **SQLNUMSTORAGEPATHS** must be set to 0 and **SQLSTORAGEPATHS** must be set to NULL.

### SQL\_AUTOMATIC\_STORAGE\_YES

The default storage group, IBMSTOGROUP, is created. 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 to create the default storage group. Currently, the choice is made based on the **SQLSTORAGEPATHS** pointer. If this pointer is NULL, the default storage group is not created, otherwise it is created. 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 58. 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.
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 <b>SQLINITSIZE</b> is being provided in bytes, kilobytes, megabytes, or gigabytes. See the information that follows this table for values.

Table 58. Fields in the SQLETSDESCEXT Structure (continued)

Field name	Data type	Description
SQLINCREASESIZEUNIT	CHAR(1)	If relevant, indicates whether <b>SQLINCREASESIZE</b> 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 <b>SQLMAXIMUMSIZE</b> 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 **SQLINITSIZEUNIT**, **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**.

## **sqledboptions 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 upgrades of this database. That is, when a database is upgraded to a subsequent DB2 release, the **UPGRADE DATABASE** 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
- IMPLICIT\_SCHEMA 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 table space 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 previously under `SQL_DB_RESTRICT_ACCESS_NO` do not occur.

**reserved**

Reserved for future use.

## sqledbterritoryinfo

This structure is used to provide code set and territory options to the sqlecrea API.

Table 59. Fields in the SQLEDBTERRITORYINFO Structure

Field Name	Data Type	Description
<b>SQLDBCODESET</b>	CHAR(9)	Database code set.
<b>SQLDBLOCALE</b>	CHAR(5)	Database territory.

### API and data structure syntax

```
SQL_STRUCTURE sqledbcountryinfo
{
    char sqldbcharset[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.
*
```

## sqleninfo

This structure returns information after a call to the sqlengne API.

Table 60. Fields in the SQLENINFO Structure

Field Name	Data Type	Description
NODENAME	CHAR(8)	The name of the node where the database is located (valid in system directory only)
LOCAL_LU	CHAR(8)	Local logical unit.
PARTNER_LU	CHAR(8)	Partner logical unit.
MODE	CHAR(8)	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	The local network adapter.
NETWORKID	CHAR(8)	Network ID.
PROTOCOL	CHAR(1)	Communications protocol.
SYM_DEST_NAME	CHAR(8)	Symbolic destination name.
SECURITY_TYPE	SMALLINT	Security type.
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.
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.

Table 60. Fields in the SLENINFO Structure (continued)

Field Name	Data Type	Description
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;
    _SQLOLDCHAR chgpwd_lu[SQL_RMTLU_SZ];
    _SQLOLDCHAR transpn[SQL_TPNAME_SZ];
    _SQLOLDCHAR lanaddr[SQL_LANADDRESS_SZ];
};
```

## COBOL Structure

```
* File: sqlenv.cbl
01 SLENINFO.
   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).
```

## sqleninfo

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.

\*



## sqlfupd

This structure passes information about database configuration files and the database manager configuration file.

Table 61. 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 in the following table:

Table 62. Updatable Database Configuration Parameters

Parameter Name	Token	Token Value	Data Type
alt_collate	SQLF_DBTN_ALT_COLLATE	809	UInt32
app_ctl_heap_sz	SQLF_DBTN_APP_CTL_HEAP_SZ	500	UInt16
appgroup_mem_sz	SQLF_DBTN_APPGROUP_MEM_SZ	800	UInt32
applheapsz	SQLF_DBTN_APPLHEAPSZ	51	UInt16
archretrydelay	SQLF_DBTN_ARCHRETRYDELAY	828	UInt16
<ul style="list-style-type: none"> <li>• auto_maint</li> <li>• auto_db_backup</li> <li>• auto_tbl_maint</li> <li>• auto_runstats</li> <li>• auto_reorg</li> </ul>	<ul style="list-style-type: none"> <li>• SQLF_DBTN_AUTO_MAINT</li> <li>• SQLF_DBTN_AUTO_DB_BACKUP</li> <li>• SQLF_DBTN_AUTO_TBL_MAINT</li> <li>• SQLF_DBTN_AUTO_RUNSTATS</li> <li>• SQLF_DBTN_AUTO_REORG</li> </ul>	<ul style="list-style-type: none"> <li>• 831</li> <li>• 833</li> <li>• 835</li> <li>• 837</li> <li>• 841</li> </ul>	UInt16
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

Table 62. Updatable Database Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
catalogcache_sz	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32
chnngpgs_thresh	SQLF_DBTN_CHNGPGS_THRESH	38	UInt16
database_memory	SQLF_DBTN_DATABASE_MEMORY	803	UInt64
dbheap	SQLF_DBTN_DB_HEAP	58	UInt64
db_mem_thresh	SQLF_DBTN_DB_MEM_THRESH	849	UInt16
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
dft_mttb_types	SQLF_DBTN_DFT_MTTB_TYPES	843	UInt32
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
discover	SQLF_DBTN_DISCOVER	308	UInt16
dlchktime	SQLF_DBTN_DLCHKTIME	9	UInt32
failarchpath	SQLF_DBTN_FAILARCHPATH	826	char(243)
groupheap_ratio	SQLF_DBTN_GROUPHEAP_RATIO	801	UInt16
hadr_local_host	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)
hadr_local_svc	SQLF_DBTN_HADR_LOCAL_SVC	812	char(41)
hadr_remote_host	SQLF_DBTN_HADR_REMOTE_HOST	813	char(256)
hadr_remote_inst	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)
hadr_remote_svc	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)
hadr_syncmode	SQLF_DBTN_HADR_SYNCMODE	817	UInt32
hadr_timeout	SQLF_DBTN_HADR_TIMEOUT	816	Sint32
indexrec	SQLF_DBTN_INDEXREC	30	UInt16
locklist	SQLF_DBTN_LOCK_LIST	704	UInt64
locktimeout	SQLF_DBTN_LOCKTIMEOUT	34	Sint16
logarchmeth1	SQLF_DBTN_LOGARCHMETH1	822	UInt16
logarchmeth2	SQLF_DBTN_LOGARCHMETH2	823	UInt16
logarchopt1	SQLF_DBTN_LOGARCHOPT1	824	char(243)
logarchopt2	SQLF_DBTN_LOGARCHOPT2	825	char(243)
logbufsz	SQLF_DBTN_LOGBUFSZ	33	UInt16
logfilsiz	SQLF_DBTN_LOGFIL_SIZ	92	UInt32
loginindexbuild	SQLF_DBTN_LOGININDEXBUILD	818	UInt32
logprimary	SQLF_DBTN_LOGPRIMARY	16	UInt16
logsecond	SQLF_DBTN_LOGSECOND	17	UInt16
max_log	SQLF_DBTN_MAX_LOG	807	UInt16
maxappls	SQLF_DBTN_MAXAPPLS	6	UInt16
maxfilop	SQLF_DBTN_MAXFILOP	3	UInt16

Table 62. Updatable Database Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
maxlocks	SQLF_DBTN_MAXLOCKS	15	UInt16
max_log	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
num_log_span	SQLF_DBTN_NUM_LOG_SPAN	808	UInt16
num_quantiles	SQLF_DBTN_NUM_QUANTILES	48	UInt16
numarchretry	SQLF_DBTN_NUMARCHRETRY	827	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
self_tuning_mem	SQLF_DBTN_SELF_TUNING_MEM	848	UInt16
seqdetect	SQLF_DBTN_SEQDETECT	41	UInt16
sheapthres_shr	SQLF_DBTN_SHEAPTHRES_SHR	802	UInt32
softmax <sup>1</sup>	SQLF_DBTN_SOFTMAX	5	UInt16
sortheap	SQLF_DBTN_SORT_HEAP	52	UInt32
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)
util_heap_sz	SQLF_DBTN_UTIL_HEAP_SZ	55	UInt32
vendoropt	SQLF_DBTN_VENDOROPT	829	char(242)

1.

**Important:** The **softmax** database configuration parameter is deprecated is deprecated in Version 10.5 and might be removed in a future release. For more information, see Some database configuration parameters are deprecated in *What's New for DB2 Version 10.5*.

The bits of SQLF\_DBTN\_AUTONOMIC\_SWITCHES indicate the default settings for a number of auto-maintenance configuration parameters. The individual bits making up this composite parameter are:

```
Default => Bit 1 on (xxxx xxxx xxxx xxx1): auto_maint
           Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
           Bit 3 on (xxxx xxxx xxxx x1xx): auto_tbl_maint
```

```

Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx0 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 0xxx xxxx): auto_storage
Bit 9 off (xxxx xxx0 xxxx xxxx): auto_stmt_stats
          0      0      0      D

```

```

Maximum => Bit 1 on  (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
Bit 8 off (xxxx xxxx 1xxx xxxx): auto_storage
Bit 9 off (xxxx xxx1 xxxx xxxx): auto_stmt_stats
          0      1      F      F

```

**Note:** The **auto\_stats\_prof** and **auto\_prof\_upd** parameters are discontinued in Version 10.5. For more information, see Some database configuration parameters are deprecated in *What's New for DB2 Version 10.5*.

Valid values for indexrec (defined in sqlutil.h):

- SQLF\_INX\_REC\_SYSTEM (0)
- SQLF\_INX\_REC\_REFERENCE (1)
- SQLF\_INX\_REC\_RESTART (2)

Table 63. 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 following note 1)
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 following note 2)

Table 63. Non-updatable Database Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16

**Note:**

1. char(17) on HP-UX, Solaris and Linux operating systems.
2. char(33) on HP-UX, Solaris and Linux operating systems.

Valid entries for the SQLFUPD token element are listed in the following table:

Table 64. 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
alternate_auth_enc	SQLF_KTN_ALTERNATE_AUTH_ENC	938	Uint16
aslheapsz	SQLF_KTN_ASLHEAPSZ	15	Uint32
audit_buf_sz	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32
authentication	SQLF_KTN_AUTHENTICATION	78	Uint16
catalog_noauth	SQLF_KTN_CATALOG_NOAUTH	314	Uint16
clnt_krb_plugin	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)
clnt_pw_plugin	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)
comm_bandwidth	SQLF_KTN_COMM_BANDWIDTH	307	float
conn_elapse	SQLF_KTN_CONN_ELAPSE	508	Uint16
cpuspeed	SQLF_KTN_CPUSPEED	42	float
dft_account_str	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)
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_timestamp	SQLF_KTN_DFT_MON_TIMESTAMP	36	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_inst	SQLF_KTN_DISCOVER_INST	308	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

Table 64. Updatable Database Manager Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
federated	SQLF_KTN_FEDERATED	604	Sint16
federated_async	SQLF_KTN_FEDERATED_ASYNC	849	Sint32
fenced_pool	SQLF_KTN_FENCED_POOL	80	Sint32
group_plugin	SQLF_KTN_GROUP_PLUGIN	810	char(33)
health_mon	SQLF_KTN_HEALTH_MON	804	Uint16
indexrec	SQLF_KTN_INDEXREC	20	Uint16
instance_memory	SQLF_KTN_INSTANCE_MEMORY	803	Uint64
intra_parallel	SQLF_KTN_INTRA_PARALLEL	306	Sint16
java_heap_sz	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32
jdk_path	SQLF_KTN_JDK_PATH	311	char(255)
keepfenced	SQLF_KTN_KEEFENCED	81	Uint16
local_gssplugin	SQLF_KTN_LOCAL_GSSPLUGIN	816	char(33)
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
mon_heap_sz	SQLF_KTN_MON_HEAP_SZ	79	Uint16
notifylevel	SQLF_KTN_NOTIFYLEVEL	605	Sint16
num_initagents	SQLF_KTN_NUM_INITAGENTS	500	Uint32
num_initfenced	SQLF_KTN_NUM_INITFENCED	601	Sint32
num_poolagents	SQLF_KTN_NUM_POOLAGENTS	502	Sint32
numdb	SQLF_KTN_NUMDB	6	Uint16
query_heap_sz	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32
resync_interval	SQLF_KTN_RESYNC_INTERVAL	68	Uint16
rqrioblk	SQLF_KTN_RQRIOBLK	1	Uint16
sheapthres	SQLF_KTN_SHEAPTHRES	21	Uint32
spm_log_file_sz	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32
spm_log_path	SQLF_KTN_SPM_LOG_PATH	313	char(226)
spm_max_resync	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32
spm_name	SQLF_KTN_SPM_NAME	92	char(8)
srvcon_auth	SQLF_KTN_SRVCON_AUTH	815	Uint16
srvcon_gssplugin_list	SQLF_KTN_SRVCON_GSSPLUGIN_LIST	814	char(256)
srv_plugin_mode	SQLF_KTN_SRV_PLUGIN_MODE	809	Uint16
srvcon_pw_plugin	SQLF_KTN_SRVCON_PW_PLUGIN	813	char(33)
ssl_cipherspecs	SQLF_KTN_SSL_CIPHERSPECS	934	char(255)
ssl_clnt_keydb	SQLF_KTN_SSL_CLNT_KEYDB	936	char(1023)
ssl_clnt_stash	SQLF_KTN_SSL_CLNT_STASH	937	char(1023)
ssl_svcename	SQLF_KTN_SSL_SVCENAME	933	char(14)

Table 64. Updatable Database Manager Configuration Parameters (continued)

Parameter Name	Token	Token Value	Data Type
ssl_svr_keydb	SQLF_KTN_SSL_SVR_KEYDB	930	char(1023)
ssl_svr_label	SQLF_KTN_SSL_SVR_LABEL	932	char(1023)
ssl_svr_stash	SQLF_KTN_SSL_SVR_STASH	931	char(1023)
ssl_versions	SQLF_KTN_SSL_VERSIONS	935	char(255)
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)
sysmon_group	SQLF_KTN_SYSMON_GROUP	808	char(30)
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
util_impact_lim	SQLF_KTN_UTIL_IMPACT_LIM	807	UInt32

**Note:** The configuration parameters **maxagents** and **maxcagents** are deprecated. In a future release, these configuration parameters may be removed completely.

Valid values for **alternate\_auth\_enc** (defined in `sqlenv.h`):

- SQL\_ALTERNATE\_AUTH\_ENC\_AES (0)
- SQL\_ALTERNATE\_AUTH\_ENC\_AES\_CMP (1)
- SQL\_ALTERNATE\_AUTH\_ENC\_NOTSPEC (255)

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\_GSSPLUGIN (9)
- SQL\_AUTHENTICATION\_GSS\_SVR\_ENC (10)
- SQL\_AUTHENTICATION\_DATAENC (11)
- SQL\_AUTHENTICATION\_DATAENC\_CMP (12)
- 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
- Bit 7 (x1xx xxxx): dft\_mon\_timestamp

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)

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 65. 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.
*
```

---

## sqllob

This structure is used to represent a LOB data type in a host programming language.

*Table 66. 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];
};
```

## 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 67. 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 <b>OBJ_NUM</b> .

Table 68. 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 <b>OBJ_TYPE</b> requires an <b>agent_id</b> (application handle). To retrieve a health snapshot with full collection information, specify SQLM_HMON_OPT_COLL_FULL in this field. <b>Note:</b> The SQLM_HMON_OPT_COLL_FULL value is deprecated and might be removed in a future release because the health monitor has been deprecated in Version 9.7.
OBJ_TYPE	INTEGER	The type of object to be monitored.
OBJECT	CHAR(128)	The name of the object to be monitored. Specified only if <b>OBJ_TYPE</b> 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 `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 `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 `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 `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 `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 `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 `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;
    _SQLOLDCHAR object[SQLM_OBJECT_SZ];
}sqlm_obj_struct;
```

**COBOL Structure**

```
* File: sqlmonct.cbl
01 SQLMA.
   05 OBJ-NUM                PIC 9(9) COMP-5.
   05 OBJ-VAR OCCURS 0 TO 100 TIMES DEPENDING ON OBJ-NUM.
       10 AGENT-ID           PIC 9(9) COMP-5.
       10 OBJ-TYPE           PIC 9(9) COMP-5.
       10 OBJECT             PIC X(128).
*
```

## 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 69. 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 70. 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 71. Fields in the SQLOPTIONS Structure

Field Name	Data Type	Description
TYPE VAL	INTEGER	Bind/precompile/rebind option type.
	INTEGER	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;
};
```

### COBOL Structure

```
* File: sql.cbl
01 SQLOPT.
   05 SQLOPTHEADER.
      10 ALLOCATED    PIC 9(9) COMP-5.
```

```
      10 USED          PIC 9(9) COMP-5.  
05 SQLOPTIONS OCCURS 1 TO 50 DEPENDING ON ALLOCATED.  
      10 SQLOPT-TYPE   PIC 9(9) COMP-5.  
      10 SQLOPT-VAL    PIC 9(9) COMP-5.  
      10 SQLOPT-VAL-PTR REDEFINES SQLOPT-VAL  
*
```

---

## SQLU\_LSN

This union 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 72. 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.

**Note:** The SQLU\_LSN structure has been replaced by the db2LSN structure.

### API and data structure syntax

```
typedef union SQLU_LSN
{
    unsigned char    lsnChar[6];
    unsigned short   lsnWord[3];
} SQLU_LSN;
```



## sqlu\_media\_list

This structure is used to pass information to the db2Load API.

Table 73. Fields in the SQLU-MEDIA-LIST Structure

Field Name	Data Type	Description
<b>MEDIA_TYPE</b>	CHAR(1)	A character indicating media type.
<b>SESSIONS</b>	INTEGER	Indicates the number of elements in the array pointed to by the target field of this structure.
<b>TARGET</b>	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 <b>media_type</b> field. For more information about what to provide in this field, see the appropriate API.
<b>FILLER</b>	CHAR(3)	Filler used for proper alignment of data structure in memory.

Table 74. Fields in the SQLU-MEDIA-LIST-TARGETS Structure

Field Name	Data Type	Description
<b>MEDIA</b>	Pointer	A pointer to an sqlu_media_entry structure.
<b>VENDOR</b>	Pointer	A pointer to an sqlu_vendor structure.
<b>LOCATION</b>	Pointer	A pointer to an sqlu_location_entry structure.
<b>PSTATEMENT</b>	Pointer	A pointer to an sqlu_statement_entry structure.
<b>PREMOTEFETCH</b>	Pointer	A pointer to an sqlu_remotefetch_entry structure.

Table 75. Fields in the SQLU-MEDIA-ENTRY Structure

Field Name	Data Type	Description
<b>RESERVE_LEN MEDIA_ENTRY</b>	INTEGER CHAR(215)	Length of the <b>media_entry</b> field. For languages other than C. Path for a backup image used by the backup and restore utilities.

Table 76. Fields in the SQLU-VENDOR Structure

Field Name	Data Type	Description
<b>RESERVE_LEN1</b>	INTEGER	Length of the <b>shr_1ib</b> field. For languages other than C.

Table 76. Fields in the SQLU-VENDOR Structure (continued)

Field Name	Data Type	Description
<b>SHR_LIB</b>	CHAR(255)	Name of a shared library supplied by vendors for storing or retrieving data.
<b>RESERVE_LEN2</b>	INTEGER	Length of the filename field. For languages other than C.
<b>FILENAME</b>	CHAR(255)	File name to identify the load input source when using a shared library.

Table 77. Fields in the SQLU-LOCATION-ENTRY Structure

Field Name	Data Type	Description
<b>RESERVE_LEN</b>	INTEGER	Length of the <b>location_entry</b> field. For languages other than C.
<b>LOCATION_ENTRY</b>	CHAR(256)	Name of input data files for the load utility.

Table 78. Fields in the SQLU-STATEMENT-ENTRY Structure

Field Name	Data Type	Description
<b>LENGTH</b>	INTEGER	Length of the data field.
<b>PDATA</b>	Pointer	Pointer to the SQL query.

Table 79. Fields in the SQLU-REMOTEFETCH-ENTRY Structure

Field Name	Data Type	Description
<b>pDatabaseName</b>	Pointer	Source Database Name.
<b>iDatabaseNameLen</b>	INTEGER	Source Database Name Length
<b>pUserID</b>	Pointer	Pointer to UserID.
<b>iUserIDLen</b>	INTEGER	UserID Length.
<b>pPassword</b>	Pointer	Pointer to Password.
<b>iPasswordLen</b>	INTEGER	Password Length.
<b>pTableSchema</b>	Pointer	Pointer to schema of source table.
<b>iTableSchemaLen</b>	INTEGER	Schema Length.
<b>pTableName</b>	Pointer	Pointer to name of source table.
<b>iTableNameLen</b>	INTEGER	Source table name Length.
<b>pStatement</b>	Pointer	Pointer to name of statement.
<b>iStatementLen</b>	INTEGER	Statement Length.
<b>pIsolationLevel</b>	Pointer	Pointer to isolation level (default CS).

Valid values for **MEDIA\_TYPE** (defined in sqlutil) are:

**SQLU\_LOCAL\_MEDIA**

Local devices (tapes, disks, or diskettes)

**SQLU\_SERVER\_LOCATION**

Server devices (tapes, disks, or diskettes; load only). Can be specified only for the **piSourceList** parameter.

**SQLU\_CLIENT\_LOCATION**

Client devices (files or named pipes). 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\_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\_list

```
} 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 *piEnableParallelism;
} 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.
*

* File: sqlutil.cbl
01 SQLU-VENDOR.
    05 SQL-SHRLIB-LEN        PIC 9(9) COMP-5.
```

```

05 SQL-SHR-LIB          PIC X(255).
05 FILLER                PIC X.
05 SQL-FILENAME-LEN     PIC 9(9) COMP-5.
05 SQL-FILENAME         PIC X(255).
05 FILLER                PIC X.
*

* File: sqlutil.cbl
01 SQLU-LOCATION-ENTRY.
   05 SQL-LOCATION-LEN     PIC 9(9) COMP-5.
   05 SQL-LOCATION-ENTRY  PIC X(255).
   05 FILLER             PIC X.
*

* File: sqlutil.cbl
01 SQLU-STATEMENT-ENTRY.
   05 SQL-STATEMENT-LEN  PIC 9(9) COMP-5.
   05 SQL-STATEMENT-ENTRY  USAGE IS POINTER.
*
```

---

## SQLU\_RLOG\_INFO

This structure contains information about the status of calls to the db2ReadLog API; and to the database log.

*Table 80. 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;
```

## sqlupi

This structure is used to store partitioning information, such as the distribution map and the distribution key of a table.

Table 81. 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 <b>SQLPARTKEY</b> 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 82. 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
Variable-length graphic string	464	465	Length in double-byte characters	Yes

Table 82. SQL Data Types and Lengths for the SQLUPI Structure (continued)

Data type	SQLTYPE (Nulls Not Allowed)	SQLTYPE (Nulls Allowed)	SQLLEN	AIX
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;
};
```



## 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 83. Fields in the SQLXA-XID Structure*

Field Name	Data Type	Description
FORMATID	INTEGER	XA format ID.
GTRID_LENGTH	INTEGER	Length of the global transaction ID.
BQUAL_LENGTH	INTEGER	Length of the branch identifier.
DATA	CHAR[128]	GTRID, followed by BQUAL and trailing blanks, for a total of 128 bytes.

**Note:** The maximum size for GTRID and BQUAL is 64 bytes each.

### 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;
```

**SQLXA\_XID**

---

## **Appendix A. Precompiler customization APIs**

### 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 about the set of Precompiler Services APIs is available from the PDF file, `prepapi.pdf`, at the following web site:

<http://www.ibm.com/software/data/db2/udb/support/manualsv9.html>

---

## Appendix B. DB2 log records

## 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 log records marked as propagatable (propagatable flag is set in the log record header) are returned when this value is used. Only propagatable log records are documented. All other log records are intended for only IBM internal use and are therefore not documented.

When a transaction invokes a log writing utility or performs writable work against a table with the `DATA CAPTURE CHANGES` attribute set to `ON`, the transaction is marked as propagatable. Only propagatable transactions have their transaction manager log records marked as propagatable.

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 record header” on page 690.

Log records are uniquely identified by a log record identifier (LRI). Because log sequence numbers (LSNs)-64-bit identifiers that determine the order of the operations that generated the log records-are generated independently on each member and there are multiple log streams, it is possible to have duplicate LSN values across different log streams. An LRI is used to identify log records across log streams; each log record in any log stream in the database is assigned a unique LRI. Use the **db2pd** command to determine which LRI is being processed by a recovery operation.

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.

*Table 84. DB2 log records*

Type	Record name	Description
Data manager	“Initialize table log record” on page 706	New permanent table creation
Data manager	“Import replace (truncate) log record” on page 707	Import replace activity
Data manager	“Activate not logged initially log record” on page 708	Alter table activity that includes the <code>ACTIVATE NOT LOGGED INITIALLY</code> clause
Data manager	“Rollback insert log record” on page 708	Rollback row insert
Data manager	“Reorg table log record” on page 709	REORG committed
Data manager	“Create index, drop index log records” on page 709	Index activity
Data manager	“Create table, drop table, rollback create table, rollback drop table log records” on page 710	Table activity

Table 84. DB2 log records (continued)

Type	Record name	Description
Data manager	"Alter table attribute log record" on page 710	Propagation, check pending, and append mode activity
Data manager	"Alter table add columns, drop columns, rollback add columns, rollback drop columns log record" on page 711	Adding columns to existing tables
Data manager	"Alter column attribute log record" on page 712	Columns activity
Data manager	"Undo alter column attribute log record" on page 712	Column activity
Data manager	"Insert record, delete record, rollback delete record, rollback update record log records" on page 713	Table record activity
Data manager	"Insert record to empty page, delete record to empty page, rollback delete record to empty page, rollback insert record to empty page log records" on page 717	Multidimensional clustered (MDC) table activity
Data manager	"Update record log record" on page 718	Row updates where storage location not changed
Data manager	"Rename of a table or schema log record" on page 719	Table or schema name activity
Data manager	"Undo rename of a table or schema log record" on page 719	Table or schema name activity
Large object (LOB) manager	"Add LOB data and add LOB amount log record structure" on page 729	LOB record activity
Long field manager	"Add/delete/non-update long field record log records" on page 700	Long field record activity
Transaction manager	"Normal commit log record" on page 693	Transaction commits
Transaction manager	"Heuristic commit log record" on page 693	Indoubt transaction commits
Transaction manager	"MPP coordinator commit log record" on page 693	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 log record" on page 694	Transaction commits. This is written on a subordinator node.
Transaction manager	"Normal abort log record" on page 695	Transaction aborts
Transaction manager	"Heuristic abort log record" on page 695	Indoubt transaction aborts
Transaction manager	"Local pending list log record" on page 695	Transaction commits with a pending list existing

Table 84. DB2 log records (continued)

Type	Record name	Description
Transaction manager	"Global pending list log record" on page 696	Transaction commits (two-phase) with a pending list existing
Transaction manager	"XA prepare log record" on page 696	XA transaction preparation in two-phase commit environments
Transaction manager	"MPP subordinator prepare log record" on page 697	MPP transaction preparation in two-phase commit environments. This log record only exists on subordinator node.
Transaction manager	"TM prepare log record" on page 698	Coordinated transaction preparation as part of a two-phase commit, where the database is acting as the TM database.
Transaction manager	"Backout free log record" on page 698	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.
Transaction manager	"Application information log record" on page 698	Information about the application that started the transaction
Transaction manager	"Federated Prepare Log Record" on page 699	Information about the federated resource manager involved in the transaction
Utility manager	"System catalog migration begin log record" on page 701	System catalog migration starts
Utility manager	"System catalog migration end log record" on page 702	System catalog migration completes
Utility manager	"Load start log record" on page 702	Table load starts
Utility manager	"Backup end log record" on page 702	Backup activity completes
Utility manager	"Table space rolled forward log record" on page 703	Table space rollforward completes
Utility manager	"Table space roll forward to point in time starts log record" on page 703	Marks the beginning of a table space rollforward to a point in time
Utility manager	"Table space roll forward to point in time ends log record" on page 703	Marks the end of a table space rollforward to a point in time
Relation manager	"DDL statement log record" on page 722	DDL activity
Relation manager	"Undo DDL statement log record" on page 726	Undo log record for DDL activity
Relation manager	"Partition information log record" on page 726	Partition activity

## Log record 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.

*Table 85. Log Manager Log Record Header (LogManagerLogRecordHeader)*

Field Description	Data Type	Offset (bytes)
Length of the log record	db2UInt32	0(4)
Type of the log record (see Table 86 on page 692)	db2UInt16	4(2)
Log record flags <sup>1</sup>	db2UInt16	6(2)
Log sequence number (LSN) of the log record	db2LSN <sup>2</sup>	8(8)
Log flush sequence (LFS) for this log record	db2UInt64	16(8)
Log sequence offset (LSO) of the previous log record in this transaction	db2UInt64	24(8)
Unique transaction identifier	SQLU_TID <sup>3</sup>	32(6)
Log stream ID	db2LogStreamIDType	38(2)
Extra log stream ID of previous LSO for compensation log records	db2LogStreamIDType	40(2)
Reserved	Not applicable	42(6)
Extra LSO for compensation log records	db2UInt64	48(8)
Extra LSO for propagatable compensation log records	db2UInt64	56(8)
<b>Note:</b> A basic log record header has a total length of 40 bytes and the log stream ID is not set. Compensation log records require an extra LSO value at the end, for a total length of 56 bytes. Propagatable compensation log records require an additional LSO value at the end, for a total length of 64 bytes. In both of these cases, the log stream ID field is set appropriately.		

**Note:**

1. Log record general flag constants

Redo always	0x0001
Propagatable	0x0002
Temp table	0x0004
Table space rollforward undo	0x0008
Singular transaction (no commit/rollback)	0x0010
Conditionally recoverable	0x0080
Table space rollforward at check constraint process	0x0100
Runtime rollback	0x0200
Pseudo compensation	0x0800

2. Log sequence number (LSN)

A 64-bit identifier that determines the order of the operations that generated the log records. The LSN is an ever-increasing value. Each member writes to its own set of log files (a log stream), and the LSN within a single log stream is a unique number.

```
db2LSN: { db2UInt64 lsnU64;
          }
```

3. Transaction identifier (TID)

A unique log record identifier representing the transaction.

```
SQLU_TID: union { unsigned char [6] ;
                  unsigned short [3] ;
                }
```

## Log record header

### 4. Record ID (RID)

A unique number identifying the position of a record.

RID: Page number char [4];

slot number char [2];

*Table 86. 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	System catalog migration begin
0x0057	System catalog migration end
0x0069	information only. A log record of type 0x0069 is an informational log record only. It is ignored by the DB2 database manager during rollforward, rollback, and crash recovery.
0x006F	Backup start
0x0071	Table Space Roll Forward to Point in Time Ends
0x007B	MPP prepare
0x007C	XA prepare
0x007D	Transaction Manager (TM) 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
0x008B	Application information

## 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 log record

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 87. Normal Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time transaction committed	sqluint64	22 (8)
Authorization identifier length <sup>1</sup> (if the log record is marked as propagatable)	unsigned short	30 (2)
Authorization identifier of the application <sup>1</sup> (if the log record is marked as propagatable)	char [ ]	32 (variable <sup>2</sup> )
Total length: 32 bytes plus variable propagatable (30 bytes nonpropagatable)		

**Note:**

1. If the log record is marked as propagatable
2. Variable based on Authorization identifier length

## Heuristic commit log record

This log record is written when an indoubt transaction is committed.

Table 88. Heuristic Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time transaction committed	sqluint64	22 (8)
Authorization identifier length <sup>1</sup> (if the log record is marked as propagatable)	unsigned short	30 (2)
Authorization identifier of the application <sup>1</sup> (if the log record is marked as propagatable)	char [ ]	32 (variable <sup>2</sup> )
Total length: 32 bytes plus variable propagatable (30 bytes nonpropagatable)		

**Note:**

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

## MPP coordinator commit log record

This log record is written on a coordinator node for an application that performs updates on at least one subordinator node.

## Transaction manager log records

Table 89. MPP Coordinator Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time transaction committed	sqluint64	22 (8)
MPP identifier of the transaction	SQLP_GXID	30 (20)
Maximum node number	unsigned short	50 (2)
TNL	unsigned char [ ]	52 (max node number/8 + 1)
Authorization identifier length <sup>1</sup> (if the log record is marked as propagatable)	unsigned short	variable (2)
Authorization identifier of the application <sup>1</sup> (if the log record is marked as propagatable)	char [ ]	variable (variable <sup>2</sup> )
Total length: variable		

**Note:**

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 log record

This log record is written on a subordinator node in MPP.

Table 90. MPP Subordinator Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time transaction committed	sqluint64	22 (8)
MPP identifier of the transaction	SQLP_GXID	30 (20)
Coordinator partition number <sup>1</sup>	unsigned short	50 (2)
Authorization identifier length (if the log record is marked as propagatable)	unsigned short	52 (2)
Authorization identifier of the application <sup>2</sup> (if the log record is marked as propagatable)	char [ ]	54 (variable <sup>3</sup> )
Total length: 54 bytes plus variable		

**Note:**

1. This is the current database partition number if the transaction is on one database partition only, otherwise it is the coordinator partition number.
2. If the log record is marked as propagatable
3. Variable based on authorization identifier length

## Normal abort log record

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 91. Normal Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Authorization identifier length <sup>1</sup> (if the log record is marked as propagatable)	unsigned short	22 (2)
Authorization identifier of the application <sup>1</sup> (if the log record is marked as propagatable)	char [ ]	24 (variable <sup>2</sup> )
Total length: 24 bytes plus variable propagatable (22 bytes nonpropagatable)		

**Note:**

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

## Heuristic abort log record

This log record is written when an indoubt transaction is aborted.

Table 92. Heuristic Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Authorization identifier length <sup>1</sup> (if the log record is marked as propagatable)	unsigned short	22 (2)
Authorization identifier of the application <sup>1</sup> (if the log record is marked as propagatable)	char [ ]	24 (variable <sup>2</sup> )
Total length: 24 bytes plus variable propagatable (22 bytes nonpropagatable)		

**Note:**

1. If the log record is marked as propagatable
2. Variable based on authorization identifier length

## Local pending list log record

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.

## Transaction manager log records

Table 93. Local Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time transaction committed	sqluint64	22 (8)
Authorization identifier length <sup>1</sup>	unsigned short	30 (2)
Authorization identifier of the application <sup>1</sup>	char [ ]	32 (variable) <sup>2</sup>
Pending list entries	variable	variable (variable)
Total Length: 32 bytes plus variables propagatable (30 bytes plus pending list entries non-propagatable)		

**Note:**

1. If the log record is marked as propagatable
2. Variable based on Authorization identifier length

### Global pending list log record

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 94. Global Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Authorization identifier length <sup>1</sup>	unsigned short	22 (2)
Authorization identifier of the application <sup>1</sup>	char [ ]	24 (variable) <sup>2</sup>
Global pending list entries	variable	variable (variable)
Total Length: 24 bytes plus variables propagatable (22 bytes plus pending list entries non-propagatable)		

**Note:**

1. If the log record is marked as propagatable
2. Variable based on Authorization identifier length

### XA prepare log record

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 re-create an indoubt transaction.

Table 95. XA Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)

Table 95. XA Prepare Log Record Structure (continued)

Description	Type	Offset (Bytes)
Time transaction prepared	sqluint64	22 (8)
Log space used by transaction	sqluint64	30 (8)
Transaction Node List Size	sqluint32	38 (4)
Transaction Node List	unsigned char [ ]	42 (variable)
Reserve	sqluint32	variable (2)
XA identifier of the transaction	SQLXA_XID <sup>1</sup>	variable (140)
Synclog information	variable	variable (variable)
Total length: 184 bytes plus variables		

**Note:** 1. For details on the SQLXA\_XID log record type, see “SQLXA\_XID” on page 683.

## MPP subordinator prepare log record

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 re-create an indoubt transaction.

Table 96. MPP Subordinator Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time Transaction Prepared	sqluint64	22 (8)
Log space used by transaction	sqluint64	30 (8)
Coordinator LSN	db2LSN	38 (8)
Padding	char [ ]	46 (2)
MPP identifier of the transaction	SQLP_GXID <sup>1</sup>	48(20)
Total Length: 68 bytes		

**Note:** 1.The SQLP-GXID log record is used to identify transactions in MPP environment.

Table 97. 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

## Transaction manager log records

### TM prepare log record

This log record is written for DB2 coordinated transactions in a single-partition database environment or on the coordinator partition 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 98. TM Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time transaction prepared	sqluint64	22 (8)
Log space used by transaction	sqluint64	30 (8)
Transaction Node List Size	sqluint32	38 (4)
Transaction Node List	unsigned char [ ]	42 (variable)
Reserve	sqluint32	variable (2)
XA identifier of the transaction	SQLXA_XID	variable (140)
Synclog information	variable	variable (variable)
Total length: 184 bytes plus variables		

### Backout free log record

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 8-byte log sequence number (*compls**sn*, stored in the log record header starting at offset 22). Under certain scenarios, the backout free log record also contains log data, starting at offset 30, 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**sn* marks the next log record to be compensated.

Table 99. Backout free Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Compls <sub>sn</sub>	db2LSN	22 (8)
Log data <sup>1</sup>	variable	variable
Total Length: 30 bytes plus variables		

**Note:** 1. Only applied in certain scenarios, and when used, the length of the entire log record in the log header is more than 28 bytes.

### Application information log record

This log record contains information about the application that started this transaction.

Table 100. Application Information Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)



Table 100. Application Information Log Record Structure (continued)

Description	Type	Offset (Bytes)
Transaction Start Time	sqluint32	22 (4)
Reserved	char[ ]	26 (16)
Code page	sqluint32	42 (4)
Application Name Length	sqluint32	46 (4)
Application Name	char [ ]	50 (variable)
Application Identifier Length	sqluint32	variable (4)
Application Identifier	char [ ]	variable (variable)
Sequence Number Length	sqluint32	variable (4)
Sequence Number	char [ ]	variable (variable)
Database Alias Used by Client Length	sqluint32	variable (4)
Database Alias Used by Client	char [ ]	variable (variable)
Authorization Identifier Length	sqluint32	variable (4)
Authorization Identifier	char [ ]	variable (variable)
<i>Total Length: 66 bytes plus variables</i>		

## Federated Prepare Log Record

This log record contains information about the federated resource managers that were involved in the transaction.

Table 101. Federated Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Number of Resource Managers	sqluint32	22 (4)
Authorization Identifier Length	sqluint16	26 (2)
Encrypted Password Length	sqluint16	28 (2)
Authorization Identifier	char [128]	30 (128)
Encrypted Password	char [263]	158 (263)
Resource Manager Entries	variable	421 (variable)
<i>Total Length: 421 bytes plus variables</i>		

## Long field manager log records

Long field manager log records are written only if a database is recoverable. That is, if the database is configured with either the **logarchmeth1** or **logarchmeth2** parameter set to a value other than OFF. The log records are written whenever long field data is inserted, deleted, or updated.

## Long field manager log records

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.

All long field manager log record offsets are from the end of the log manager log record header.

When a table was altered to capture LONG VARCHAR OR LONG VARGRAPHIC columns by specifying INCLUDE LONGVAR COLUMNS on the ALTER TABLE statement:

- The long field manager writes 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 a delete or 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 102. Long field manager log record header (LongFieldLogRecordHeader)*

Description	Type	Offset (Bytes)
Originator code (component identifier = 3)	unsigned char	0 (1)
Operation type (See Table 103.)	unsigned char	1 (1)
Table space identifier	unsigned short	2 (2)
Object identifier	unsigned short	4 (2)
Parent table space identifier <sup>1</sup>	unsigned short	6 (2)
Parent object identifier <sup>2</sup>	unsigned short	8 (2)
<i>Total Length: 10 bytes</i>		

### Note:

1. Table space ID of the data object
2. Object ID of the data object

*Table 103. 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 log records

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 104. Add/Delete/Non-update long field record log record structure

Description	Type	Offset (Bytes)
Log header	LongFieldLogRecordHeader	0 (10)
Internal	Internal	10 (1)
Original operation type <sup>1</sup>	char	11 (1)
Column identifier <sup>2</sup>	unsigned short	12 (2)
Long field length <sup>3</sup>	unsigned short	14 (2)
File offset <sup>4</sup>	sqluint32	16 (4)
Long field data	char[ ]	20 (variable)

**Note:**

## 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.

**Utility manager log records**

The utility manager produces log records associated with the following DB2 utilities: database upgrade; load; backup; table space rollforward.

The log records signify the beginning or the end of the requested activity. Only propagatable log records for these utilities are documented.

**System catalog migration begin log record**

During database upgrade, the system catalog objects are converted to the new release format. This log record indicates the start of system catalog migration.

Table 105. System catalog migration Begin Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Start time	char[ ]	22 (10)
Previous release	unsigned short	32 (2)
New release	unsigned short	34 (2)
<i>Total Length: 36 bytes</i>		

## System catalog migration end log record

During database upgrade, the system catalog objects are converted to the new release format. This log record indicates the successful completion of system catalog migration.

*Table 106. System catalog migration End Log Record Structure*

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
End time	char[ ]	22 (10)
New release	unsigned short	32 (2)
<i>Total Length: 34 bytes</i>		

## Load start log record

This log record is associated with the beginning of a load.

It is the only Load log record that is propagatable.

For the purpose of log record propagation, it is recommended that after reading a Log Start log record you not continue to propagate log records for the specific 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 107. Load Start Log Record Structure*

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Log record identifier	squint32	22 (4)
Pool identifier	unsigned short	26 (2)
Object identifier	unsigned short	28 (2)
Flag	squint32	30 (4)
Object pool list	variable	34 (variable)
<i>Total length: 34 bytes plus variable</i>		

## Backup end log record

This log record is associated with the end of a successful backup.

*Table 108. Backup End Log Record Structure*

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Backup end time	squint64	22 (8)
<i>Total Length: 30 bytes</i>		

## Table space rolled forward log record

This log record is associated with table space **ROLLFORWARD** recovery. It is written for each table space that is successfully rolled forward.

*Table 109. Table Space Rolled Forward Log Record Structure*

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Table space identifier	sqluint32	22 (4)
<i>Total length: 26 bytes</i>		

## Table space roll forward to point in time starts log record

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 110. Table Space Roll Forward to Point in Time Starts Log Record Structure*

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time stamp for this log record	sqluint64	22 (8)
Time stamp to which table spaces are being rolled forward	sqluint32	30 (4)
Number of pools being rolled forward	sqluint32	34 (4)
<i>Total length: 38 bytes</i>		

## Table space roll forward to point in time ends log record

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 111. Table Space Roll Forward to Point in Time Ends Log Record Structure*

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0 (22)
Time stamp for this log record	sqluint64	22 (8)
Time stamp to which table spaces were rolled forward	sqluint32	30 (4)
A flag whose value is TRUE if the roll forward was successful, or FALSE if the roll forward was cancelled.	sqluint32	34 (4)
<i>Total length: 38 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

## Utility manager log records

second, to understand the ordering of events it is necessary to have further precision. The second timestamp field provides 4 bytes that are used as a counter to indicate the ordering of the log records that occur within the same second. If the log record timestamps of two log records are identical, the additional 4 byte timestamp counter field can be used to determine the ordering of the associated log events.

## 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 112. DMS log record header structure (DMSLogRecordHeader)*

Description	Type	Offset (Bytes)
Component identifier (=1)	unsigned char	0(1)
Function identifier (See Table 113.)	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 113. DMS log record header structure function identifier values and definitions*

Value	Definition
102	Add columns to table
104	Undo add columns
108	Undo update partition state
110	Undo insert record
111	Undo delete record
112	Undo update record
113	Alter column
115	Undo alter column
116	Alter column information for a LOB column
119	Undo alter column information for a LOB column
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
137	Update partition state
161	Delete record

Table 113. DMS log record header structure function identifier values and definitions (continued)

Value	Definition
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
169	Drop columns
170	Undo drop columns
171	Changing row version operation
172	Undo changing row version operation

Table 114. DOM log record header structure (DOMLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=4)	unsigned char	0(1)
Function identifier (See Table 115.)	unsigned char	1(1)
Object identifiers		
Table space identifier	unsigned short	2(2)
Object identifier	unsigned short	4(2)
Table identifiers		
Table space identifier	unsigned short	6(2)
Table identifier	unsigned short	8(2)
Object type	unsigned char	10(1)
Flags	unsigned char	11(1)
Total Length: 12 bytes		

Table 115. 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

## Data manager log records

**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 log record

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. The function ID is 128.

Table 116. Initialize table log record structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
File create LSN	db2LSN	6(8)
Internal	Internal	14(74)
Table description length	sqluint32	88(4)
Table description record	variable	92(variable)
Total Length: 92 bytes plus table description record length		

Table 117. Table description record

Description	Type	Offset (Bytes)
record type	unsigned char	0(1)
Internal	Internal	1(1)
number of columns	unsigned short	2(2)
array of column descriptor	variable long	variable
Total length: 4 bytes plus the array of column descriptor length		

#### 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
DECFLOAT64	0x0006
DECFLOAT128	0x0007
CHAR	0x0100
VARCHAR	0x0101



LONG VARCHAR	0x0104
DATE	0x0105
TIME	0x0106
TIMESTAMP	0x0107
BLOB	0x0108
CLOB	0x0109
STRUCT	0x010D
XMLTYPE	0x0112
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 0, unsigned char, precision (total length) Byte 1, 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

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)
- inline length (INLINE\_LENGTH, sqluint16, 2 bytes)
- log flag (IS COLUMN LOGGED, sqluint16, 2 bytes)
- reserved (internal, sqluint32. 4 bytes)

The first LOB, CLOB, or DBCLOB encountered in the column descriptor array uses the first element in the LOB descriptor array. The second LOB, CLOB, or DBCLOB encountered in the column descriptor array uses the second element in the LOB descriptor array, and so on.

#### Import replace (truncate) log record

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. The function ID is 11.

## Data manager log records

Table 118. Import replace (truncate) log record structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	Internal	12(variable)
Total Length: 12 bytes plus variable length		

### Activate not logged initially log record

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. This is function ID 12.

Table 119. Active not logged initially log record structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	Internal	12(4)
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)
LOB Object ID	unsigned short	24(2)
XML Object ID	unsigned short	26(2)
Total Length: 28 bytes		

\* Same as table space identifiers in the DOM header; it is a unique identifier for each table space defined in the database.

### Rollback insert log record

The rollback insert log record is written when an insert row action (INSERT RECORD) is rolled back. This is a Compensation log record. The function ID is 110.

Table 120. Rollback insert log record structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Internal	Internal	6(2)
Record Length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Total Length: 16 bytes		

### IrlUDflags

Table 121. IrlUDflags values and definition

Value	Definition
0x0200	The operation is the result of redistribute or load

Table 121. IrlUDflags values and definition (continued)

Value	Definition
0x0400	The operation is the result of internal temporal operation
0x0800	The operation is the result of an MQT maintenance operation
0x1000	The operation is the result of a referential integrity operation
0x2000	The operation is the result of a trigger
0x4000	The operation is the result of a SET INTEGRITY statement
0x8000	The operation is the result of a decomposed update

## Reorg table log record

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. The function ID is 35.

Table 122. Reorg table log record structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(476)
Index token <sup>1</sup>	unsigned short	488(2)
Temporary table space ID <sup>2</sup>	unsigned short	490(2)
Long temporary table space ID	unsigned short	492(2)
<i>Total Length: 494 bytes</i>		

### Note:

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
- The index token, which is equivalent to the IID column in SYSCAT.INDEXES. 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. The function ID is either 2 (create index) or 3 (drop index).

Table 123. Create index, drop index log records structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	Internal	12(2)
Index token	unsigned short	14(2)
Index root page	sqluint32	16(4)
Total Length: 20 bytes		

### Create table, drop table, rollback create table, rollback drop table log records

These log records are written when the DATA object for a permanent table is created or dropped. For creation of an MDC or ITC 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 before table initialization (Initialize Table). Create table and drop table are normal log records. Rollback create table and rollback drop table are Compensation log records. The function ID is either 101 (create table), 4 (drop table), 130 (rollback create table), or 5 (rollback drop table).

Table 124. Create table, drop table, rollback create table, rollback drop table log records structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(72)
Total Length: 84 bytes		

### Alter table attribute log record

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. The function ID is 124.

Table 125. Alter table attribute, undo alter table attribute

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Alter bit (attribute) mask	sqluint64	6(8)
Alter bit (attribute) values	sqluint64	14(8)
Total Length: 22 bytes		

#### 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 previously listed bits 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, drop columns, rollback add columns, rollback drop columns log record

These log files are written when the user is adding or dropping 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 before 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.

Alter table add columns and drop columns are Normal log records. Rollback add columns and rollback drop columns are Compensation log record. The function IDs are 102 (add column), 104 (undo add column), 169 (drop columns) or 170 (undo drop columns).

*Table 126. Alter table add columns, drop columns, rollback add columns, rollback drop columns log records structure*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordheader	0(6)
Internal	Internal	6(2)
Old column count	sqluint32	8(4)
New column count	sqluint32	12(4)
Old parallel arrays <sup>1</sup>	variable	16(variable)
New parallel arrays	variable	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 the description following Table 117 on page 706.

### Change table structure version, rollback change table structure version log records

These log records are written after the column definitions of a table that has been changed by an ALTER TABLE statement.

## Data manager log records

Changing table structure version is a Normal log record. Rollback changing table structure version is a Compensation log record. The function IDs are 171 (changing table structure version) or 172 (undo changing table structure version).

*Table 127. Changing table structure version log record structure*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Number of version mappings logged	unsigned short	6(2)
Offset to additional data	sqlint32	8(4)
Previous table version number	unsigned char	12(1)
Current table version number	unsigned char	13(1)
Internal	Internal	14(2)
Array of offsets into additional data	variable	16(variable)
Additional data	variable	variable(variable)
Total length: 16 bytes plus variable length		

*Table 128. Undo changing table structure version log record structure*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Internal	Internal	6(variable)
Total length: 6 bytes plus variable length		

## Alter column attribute log record

The function IDs are 113 for non-LOB columns and 116 for LOB columns.

*Table 129. Alter column attribute log record structure*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordheader	0(6)
Column ID	unsigned short	6(2)
Old column definition	Column descriptor <sup>1</sup>	8(8)
New column definition	Column descriptor <sup>1</sup>	16(8)
Total Length: 24 bytes plus record length.		

<sup>1</sup> For a description of the column descriptor array, see the description following Table 117 on page 706.

## Undo alter column attribute log record

The function IDs are 115 for non-LOB columns and 119 for LOB columns.

*Table 130. Undo alter column attribute log record structure*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)

Table 130. Undo alter column attribute log record structure (continued)

Description	Type	Offset (Bytes)
Column ID	unsigned short	6(2)
Old column definition	Column descriptor <sup>1</sup>	8(8)
New column definition	Column descriptor <sup>1</sup>	16(8)
<i>Total Length: 24 bytes plus record length.</i>		

<sup>1</sup> For a description of the column descriptor array, see the description following Table 117 on page 706.

### Insert record, delete record, rollback delete record, rollback update record log records

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. The function IDs are 162 (insert), 161 (delete), 111 (rollback delete), or 112 (rollback update).

Table 131. Insert record, delete record, rollback delete record, rollback update record log record structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
lrIUDflags	unsigned short	6(2)
Record Length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Record offset	unsigned short	18(2)
Record header and data	variable	20(variable)
<i>Total Length: 20 bytes plus record length</i>		

Following are details about the record header and data:

#### 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 having the 1 bit set:

## Data manager log records

- Fixed length (unsigned short, 2 bytes). This is the length of the fixed length section of the data row.
  - Formatted record (all of the fixed length columns, followed by the variable length columns).
  - The following fields apply to user data records with record type having the 2 bit set:
    - Number of columns (unsigned short, 2 bytes). This is the number of columns in the data portion of the data row. See “Formatted user data record for table with VALUE COMPRESSION” on page 716.
- Note:** the offset array will contain 1 + the number of columns.
- Formatted record (offset array, followed by the data columns).

Table 132. Rollback delete record and rollback update record log record structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Internal	Internal	6(2)
Record Length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Record offset	unsigned short	18(2)
Record header and data	variable	20(variable)
Total Length: 20 bytes plus record length		

A user record is specified completely by the following characteristics:

1. Outer record type is 0, or
2. Outer record type is 0x10, or
3. Outer record type has the 0x04 bit set and
  - Inner record type has the 0x01 bit set, or
  - Inner record type has the 0x02 bit set.

### Extracting inlined LOB data from formatted user data record

Inlined LOB data can be extracted from the user data record. Once the beginning of the LOB column data is located (based on whether the table has VALUE COMPRESSION enabled or not), examination of the first byte can reveal the presence of inlined LOB data.

If the first byte is 0x69, it marks the beginning of a 4-byte inlined LOB data header. The inlined LOB data begins after this 4-byte header.

If the first byte is 0x80, the LOB data is an empty string.

### 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
- BLOB
- CLOB
- VARGRAPHIC
- LONG VARG
- DBCLOB

The length of the fixed portion of the different field types can be determined as follows:

#### DECIMAL

This field is a standard packed decimal in the form: *nnnnnnn...s*. The length of the field is:  $(\text{precision} + 2)/2$ . 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 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 descriptor record structure

- record type
- number of columns
- column structure
  - field type
  - length
  - null flag
  - field offset
- LOB information

**Note:** For more information, see the description following Table 116 on page 706.

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:

## Data manager log records

- 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.

### 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 the description following Table 131 on page 713.

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. If the high bit (0x8000) of the offset is set and the data record is varying-length string data, then the data is stored out of row. Clearing the high bit returns the offset for the varying-length data descriptor and in this case the length field is 24.

### 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.
5. For varying-length string data and if the high bit (0x8000) in the offset is used to indicate the accessed data is a varying-length data descriptor and in this case the length field is 24. The varying-length data is stored out of row.

**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 the description following Table 131 on page 713.

**Insert record to empty page, delete record to empty page, rollback delete record to empty page, rollback insert record to empty page log records**

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 the 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 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. The function IDs are 165 (insert record to empty page), 164 (delete record to empty page), 166 (rollback delete record to empty page), or 131 (rollback insert record to empty page).

*Table 133. Rollback insert record to empty page*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
lrIUDflags	unsigned short	6(2)
Record length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Internal	Internal	18(2)
First page of the block	sqluint32	20(4)
<i>Total length: 24 bytes</i>		

*Table 134. Insert record to empty page*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
lrIUDflags	unsigned short	6(2)
Record Length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Internal	Internal	18(2)
First page of the block	sqluint32	20(4)
Record offset	unsigned short	24(2)

## Data manager log records

Table 134. Insert record to empty page (continued)

Description	Type	Offset (Bytes)
Record header and data	variable	26(variable)
Total Length: 26 bytes plus Record length		

**Note:** For Record Header and Data Details, see the description following Table 131 on page 713.

Table 135. Rollback delete record to empty page and delete record to empty page

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Internal	Internal	6(2)
Record Length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Internal	Internal	18(2)
First page of the block	squint32	20(4)
Record offset	unsigned short	24(2)
Record header and data	variable	26(variable)
Total Length: 26 bytes plus Record length		

## Update record log 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, delete record, rollback delete record, rollback update record log records” on page 713). 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. The function ID is 163.

Table 136. Update record log record structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
lrUDflags	unsigned short	6(2)
New Record Length	unsigned short	8(2)
Free space	unsigned short	10(2)
RID	char[]	12(6)
Record offset	unsigned short	18(2)
Old record header and data	variable	20(variable)
Log header	DMSLogRecordHeader	variable(6)
Internal	Internal	variable(2)
Old Record Length	unsigned short	variable(2)
Free space	unsigned short	variable(2)
RID	char[]	variable(6)

Table 136. Update record log record structure (continued)

Description	Type	Offset (Bytes)
Record offset	unsigned short	variable(2)
New record header and data	variable	variable(variable)
<i>Total Length: 40 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. This is function ID 122.

Table 137. 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. This is function ID 123.

Table 138. Undo 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.

## Insert multiple records, undo insert multiple records

These log records are written when multiple rows are inserted into the same page of a table. Rollback insert multiple record is a compensation log record. The function IDs are 167 and 168.

Table 139. Insert multiple records structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[]	6(2)
Number of records	unsigned short	8(2)
Free space	unsigned short	10(2)
Sum of record lengths	unsigned short	12(2)

## Data manager log records

Table 139. Insert multiple records structure (continued)

Description	Type	Offset (Bytes)
Variable part length	unsigned short	14(2)
Pool page number	sqluint32	16(4)
Record descriptions or rollback descriptions	variable	20(variable)
See Table 140 and Table 141.		
Total Length: 20 bytes plus record length		

Table 140. Records descriptions (one for each record)

Description	Type	Offset (Bytes)
RID	unsigned char[6]	0(6)
Record offset	unsigned short	6(2)
Record header and data	variable	8(variable)
Total Length: 8 bytes plus record length		

Table 141. Rollback descriptions (one for each record)

Description	Type	Offset (Bytes)
RID	unsigned char[6]	0(6)
Record offset	unsigned short	6(2)
Total Length: 8 bytes		

For record header and data details, see the description following Table 131 on page 713.

## Update partition state, rollback partition state log records

The update partition state log record is written when a user issues an ALTER TABLE statement with ADD PARTITION, ATTACH PARTITION, or DETACH PARTITION clauses. It is also written when the SET INTEGRITY statement is executed on a partitioned table to bring a previously attached partition online and visible. The log record function ID is 137, while the undo or rollback log record function ID is 108.

Table 142. Update partition state, rollback partition state log record structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Master table space identifier	unsigned short	6(2)
Master table identifier	unsigned short	8(2)
Internal	Internal	10(2)
Data partition identifier	unsigned short	12(2)
Internal	Internal	14(6)
Internal	Internal	20(2)
Partition action	unsigned short	22(2)
Total Length: 24 bytes		

**Log header**

See Table 112 on page 704. The table space and table identifiers in the DMSLogRecordHeader match the TBSPACEID and PARTITIONOBJECTID column values in the SYSCAT.DATAPARTITIONS catalog view for the table partition.

**Master table space identifier**

The master table space identifier matches the TBSPACEID column value in the SYSCAT.TABLES catalog view for the partitioned table.

**Master table identifier**

The master table identifier matches the TABLEID column value in the SYSCAT.TABLES catalog view for the partitioned table.

**Data partition identifier**

The data partition identifier matches the DATAPARTITIONID column value in the SYSCAT.DATAPARTITIONS catalog view for the table partition.

**Partition action**

The partition action values have the following definitions:

*Table 143. Data partition action values and definitions:*

Action value	Definition
1	ADD PARTITION
2	ATTACH PARTITION
4	SET INTEGRITY after ATTACH PARTITION
5	DETACH PARTITION (Deferred: Materialized query tables (MQT) need to be maintained.) <sup>1</sup>
6	DETACH PARTITION (Deferred: Nonpartitioned indexes require cleanup.) <sup>1</sup>
7	DETACH PARTITION (Immediate: Attached partition never had SET INTEGRITY executed.) <sup>1</sup>
8	DETACH PARTITION (Immediate: There are no MQT or nonpartitioned indexes.) <sup>1</sup>
13	DETACH PARTITION (Deferred: Logically detached) <sup>2</sup>
15	DETACH PARTITION (Deferred: Materialized query tables (MQT) need to be maintained.) <sup>2</sup>
16	DETACH PARTITION (Deferred: Attached partition never had SET INTEGRITY executed.) <sup>2</sup>
18	DETACH PARTITION (Deferred: Detached dependents maintained) <sup>2</sup>
20	DETACH PARTITION (Deferred: Nonpartitioned indexes require cleanup.) <sup>2</sup>
21	DETACH PARTITION (Immediate: DETACH complete.) <sup>2</sup>

**Note:**

1. Starting with DB2 Version 9.7 Fix Pack 1, the action value will not be generated. The value is listed for backward compatibility only.
2. The action value is defined starting with DB2 Version 9.7 Fix Pack 1.

## Relation manager log records

Relation manager log records facilitate the replication of DDL operations to a target database.

Table 144. Relation manager log record header structure (*RDSLogRecordHeader*)

Description	Type	Offset (Bytes)
Component identifier (=11)	unsigned char	0(1)
Function identifier (See Table 145.)	unsigned char	1(1)
Table identifiers		
Table space identifier	unsigned short	2(2)
Table identifier	unsigned short	4(2)
rdsLogRecFlags	unsigned short	6(2)
<i>Total Length: 8 bytes</i>		

Table 145. Relation manager log record header structure function identifier values and definitions

Value	Definition
4	DDL statement
5	Undo DDL statement
6	Partition information

### DDL statement log record

DDL statement log records are written for all Data Definition Language (DDL) operations. They include high-level information for the DDL, such as the type of operation (for example, CREATE, ALTER, or DROP) as well as the type of object (for example, TABLE, VIEW, or INDEX). There is also some information related to the compilation environment in which the DDL statement is executed, including the default schema name, function path, and statement authorization id.

DDL statement log records contain the actual SQL statement for the DDL operation, and if the statement is too long to fit in one log record, the statement will be split into multiple log records, each of which includes the total length of the statement and the length of the remaining text.

The function ID is 4.

Table 146. DDL statement log record structure

Description	Type	Offset (Bytes)
Log header	RDSLogRecordHeader	0(8)
Operation identifier (See Table 147 on page 723)	unsigned short	8(2)
Object identifier (See Table 148 on page 723)	unsigned short	10(2)
Options	unsigned short	12(2)
NumEntries	unsigned short	14(2)
DDL statement text left	squint32	variable(4)



*Table 146. DDL statement log record structure (continued)*

Description	Type	Offset (Bytes)
DDL statement text length	sqluint32	variable(4)
Entries Each entry (See Table 149 on page 726)	DDLStmtRecordEntry	16(NumEntries * 4)
Entry text Each entry text length determined by the corresponding length in DDLStmtRecordEntry	char[]	variable(variable)
DDL statement text	char[]	variable(variable)

*Table 147. DDL operation identifier values and definitions*

Value	Definition
1	CREATE
2	DROP
3	ALTER
4	SET CONSTRAINTS
5	RENAME
6	COLLECT STATISTICS
7	ATTACH
8	DETACH
9	FLUSH
10	COMMENT
11	GRANT
12	REVOKE
13	LOCK
14	TRUNCATE
15	TRANSFER
16	RECREATE
17	SET
18	REFRESH

*Table 148. DDL object identifier values and definitions*

Value	Definition
1	ALIAS
2	CHECK CONSTRAINT
3	COLUMN
4	DATABASE
5	FUNCTION
6	INDEX
7	MONITOR

## Relation manager log records

Table 148. DDL object identifier values and definitions (continued)

Value	Definition
8	PACKAGE
9	REFERENTIAL INTEGRITY CONSTRAINT
10	TABLE
11	TABLE SPACE
12	TRIGGER
13	TYPE
14	VIEW
15	CONSTRAINT
16	SCHEMA
17	PARTITION GROUP
18	BUFFERPOOL
19	PROCEDURE
20	DISTINCT TYPE
21	ABSTRACT DATA TYPE
22	INDEX EXTENSION
23	INDEX METHOD
24	NICKNAME
25	FUNCTION MAPPING
26	TYPE MAPPING
27	STORED PROCEDURE NICKNAME
28	SERVER MAPPING
29	USER MAPPING
30	SERVER OPTION
31	REVERSE TYPE MAPPING
32	WRAPPER
33	PASS THROUGH
34	COMPOUND
35	TABLE HIERARCHY
36	VIEW HIERARCHY
37	METHOD DEFINITION
38	COLLECT STATS
39	METHOD BODY
40	SEQUENCE
41	XML CONTAINER
42	XML COLLECTION
43	XML COLLECTION VIEW
44	XML ITEM VIEW
45	XML INDEX
46	XSR OBJECT
47	SECURITY LABEL COMPONENT

Table 148. DDL object identifier values and definitions (continued)

Value	Definition
48	SECURITY LABEL
49	SECURITY LABEL POLICY
50	ROLE
51	VARIABLE
52	WORKLOAD
53	SERVICE CLASS
54	WORK CLASS SET
55	WORK ACTION SET
56	THRESHOLD
57	HISTOGRAM TEMPLATE
58	TRUSTED CONTEXT
59	AUDIT POLICY
60	MODULE
61	MODULE BODY
62	PERMISSION
63	MASK
64	PROFILE CACHE
65	AUTHORITIES AND PRIVILEGES
66	EXEMPTION
67	SUMMARY TABLE
68	GLOBAL TEMP TABLE
69	CREATED TEMPORARY TABLE
70	INSERT ONLY TABLE
71	SUBTABLE
72	TABLE OF STRUCTURED TYPE
73	STRUCTURED TYPE
74	ROW TYPE
75	TABLE ALIAS
76	SEQUENCE ALIAS
77	MODULE ALIAS
78	PRIVATE ALIAS
79	PUBLIC ALIAS
80	EVENT MONITOR
81	SERVER
82	TRANSFORM
83	STORAGE GROUP
84	USAGE LIST
85	FEDERATED CACHE

## Relation manager log records

Table 149. *DDLStmtRecordEntry* structure

Description	Type	Offset (Bytes)
Entry type (See Table 150)	unsigned short	0(2)
Entry length	unsigned short	2(2)

Table 150. *Entry type values and definitions*

Value	Definition
1	Default schema name
2	Function path
3	Authorization ID

### Undo DDL statement log record

The function ID is 5.

Table 151. *Undo DDL Statement log record structure*

Description	Type	Offset (Bytes)
Log header	RDSLogRecordHeader	0(8)

### Partition information log record

Partition information log records are added when you create a partitioned table, as well as when you add, attach, or detach a partition. There is one such log record per each partition. The function ID is 6.

Table 152. *Partition Information log record structure*

Description	Type	Offset (Bytes)
Log header	RDSLogRecordHeader	0(8)
Partition action (See Table 153 on page 727)	unsigned short	8(2)
Partition range flags (See Table 154 on page 727)	unsigned short	10(2)
Partition ID	unsigned short	12(2)
Data tablespace ID	unsigned short	14(2)
Long tablespace ID	unsigned short	16(2)
Index tablespace ID	unsigned short	18(2)
Partition name length	unsigned short	20(2)
Table schema length	unsigned short	22(2)
Table name length	unsigned short	24(2)
Low value length	unsigned short	26(2)
High value length	unsigned short	28(2)
Partition name	char[]	30(variable)
Table schema	char[]	variable(variable)
Table name	char[]	variable(variable)
Low value <sup>1</sup>	char[]	variable(variable)

Table 152. Partition Information log record structure (continued)

Description	Type	Offset (Bytes)
High value <sup>1</sup>	char[]	variable(variable)
<sup>1</sup> The low and high values are logged as an ASCII string. For example, even if the column type is integer and the value is 100, in the log record, the value will be a three-byte string "100"..		

Table 153. Partition action values and definitions

Value	Definition
1	CREATE
2	ADD
3	ATTACH
4	DETACH

Table 154. Partition range flags values and definitions

Value	Definition
0x0001	Low value inclusive
0x0002	High value inclusive

## LOB manager log records

The large object (LOB) manager marks LOB update and insert log records as propagatable when the table has DATA CAPTURE CHANGES enabled. Prior to Version 10.1, these log records were only written to the log stream but they were never marked as propagatable. The information in LOB manager log records can be used by replication solutions to replicate LOB columns.

Update and insert LOB log records appear before the DMS row log record, which makes it difficult for replication solutions to track and apply the changes. As a result, there is a DMS log record that marks the beginning of these out of row data log records. This DMS log record is an UNDO only log record; when it is rolled back, an informational compensation log record is written out. The compensation log record is also marked as propagatable

All LOB manager log records begin with a header. When a table has been altered to enable DATA CAPTURE CHANGES, the following things occur:

- The LOB manager writes the appropriate LOB log record.
- When LOB data is updated, one of two things happen:
  - If the update is not a concatenation, the update is treated as a delete of the old LOB value, followed by an insert of the new LOB value; however, the delete operation is not propagated. To determine whether or not an add LOB data record is associated with an update operation on the table, the original operation value is logged to the LOB manager log record.
  - If the update is a concatenation, the LOB manager logs this operation by logging an add LOB data log record of the data being concatenated to the LOB data.
- The LOB manager might write multiple add LOB data log records for one insert or update operation. This can happen in the following cases:
  - The column was defined with the COMPACT option
  - The LOB data being inserted is greater than 32 KB in length
- The LOB manager never writes log records for zero length LOB data that is being insert or updated. If the LOB data can be inlined, then the LOB data is stored in the DMS row, so no LOB log record is logged in that case.
- If the NOT LOGGED option is specified for a LOB column, an Add LOB amount record is logged, and propagated, in place of each Add LOB data record that would have been logged for that particular column.

### LOB manager log record header (LOBLogRecordHeader)

Table 155. LOB manager log record header (LOBLogRecordHeader)

Description	Type	Offset (Bytes)
Originator code (component identifier = 5)	unsigned char	0 (1)
Operation type	unsigned char	1 (1)
Table space identifier	unsigned short	2 (2)
Object identifier	unsigned short	4 (2)
Parent table space identifier	unsigned short	6 (2)
Parent object identifier	unsigned short	8 (2)
Internal	Internal	10 (2)

The total length of the LOB manager log record header is 12 bytes.

## LOB manager log record header operation type values and definitions

Table 156. LOB manager log record header operation type values and definitions

Value	Definition
64	Add LOB data record
65	Add LOB amount record
66	Delete LOB data record (information only)
67	Non-update LOB data record (information only)

## Add LOB data and add LOB amount log record structure

Table 157. Add LOB data and add LOB amount log record structure

Description	Type	Offset (Bytes)
Log header	LOBLogRecordHeader	0 (12)
LOB length <sup>1</sup>	sqluint32	12 (4)
Byte offset <sup>2</sup>	sqluint64	16 (8)
Internal	Internal	24 (1)
Original operation type <sup>3</sup>	unsigned char	25 (1)
Column identifier <sup>4</sup>	unsigned short	26 (2)
Internal	Internal	28 (4)
LOB data <sup>5</sup>	char[]	32 (variable)

<sup>1</sup> LOB data length in bytes.

<sup>2</sup> Byte offset into LOB data object where data is to be located.

<sup>3</sup> Original operation type:

- 1: Insert
- 2: Delete
- 4: Update
- 8: Concat

<sup>4</sup> The column number that the log record is applied to. The column number starts at 0. All out of row varying-length string data that is stored in a LOB data object is consolidated into a single data object and the column number is set to 65535 (see the following section for more information).<sup>5</sup> The add LOB amount log record does not contain any actual LOB data, so you should ignore this field.

### Log records with column number 65535:

LOB log records with a column number of 65535 have the following behaviors:

- When a row is inserted and part of its varying-length string data is stored as LOB data, the LOB manager writes the appropriate LOB data log record with a column identifier of 65535 and the original operation type in the log record set to 1. This log record is written before the data manager insert log record.
- When a row is deleted, the old LOB data (containing the old string data) is deleted and is logged if DATA CAPTURE is ON. The original operation type in the log record is set to 2 and the column identifier is set to 65535. This log record is written after the data manager delete log record.
- When a row with part of its varying-length string data stored as LOB data has its out of row varying-length string data updated, the update is treated as an

## LOB manager log records

insert and a delete. The old LOB data is deleted and new LOB data might be inserted, if the updated row contains out of row varying-length string data. Both log records have the original operation type set to 4 and the column identifier is set to 65535. The log records are written before the data manager update log record.

- When a row with part of its varying-length string data stored as LOB data is updated, but there is no change to the out of row varying length string data, a non-update LOB data record is written before the data manager update log record.
- The delete LOB data record and the non-update LOB data record are information only log records.

If a LOB data object is too big to be logged, it can be split and logged by multiple LOB data log records. In such cases, you can locate all the LOB data log records with the same column identifier, operation type, and original operation type and then concatenate all the LOB data from each LOB data log record.

When the column identifier is 65535, the corresponding LOB data (after applying concatenation, if necessary) is the consolidation of all the out of row varying-length string data. The structure to retrieve individual varying-length string data consists of the following:

### Header

The first four bytes is the header for the structure.

- The first byte has an "eye-catcher" value of 0x12.
- The next 3 bytes indicate the size of the structure (including header) in big endian format.

### Offset array and data

An offset array and the data portion follow the header. Each entry in the offset array is a 4-byte offset to the corresponding column data in the data portion. The number of entries in the offset array is one plus the number of table columns at the time when this log record was written.

- For a varying-length string data with column identifier 'n' (column identifiers start at 0), you can look up its offset from the offset array.
- The length of the string data is the difference between the current offset and the offset of the next column.
- Only out of row varying-length string data has a non-zero length.

## Start of out of row data log record

The start of out of row data log record is written when long field (LF) or LOB data might be logged for a table which has DATA CAPTURE CHANGES enabled. This function ID for this log record is 211.

*Table 158. Start of out of row data log record*

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0 (6)

The total length of the start of out of row data log record is 6 bytes.

**Note:** It is possible there might not be any LF or LOB log records following the start of out of row data log record.



## Undo start of out of row data log record

The function ID for this log record is 211.

Table 159. Undo start of out of row data log record

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0 (6)

The total length of the start of out of row data log record is 6 bytes.

## Log flow examples

The log record sequence for the following examples flow from top to bottom, and only propagated log records are shown.

Assume the table DDL is defined as follows:

```
CREATE TABLE T1 (C1 INT, C2 CHAR(30), C3 CLOB, C4 CLOB) DATA CAPTURE CHANGES
```

- If you insert a row into T1, the sequence of log records is as follows:
  1. Start of the out of row data log record
  2. Add LOB data log record (for column C3)
  3. Add LOB data log record (for column C4)
  4. DMS insert record log record (the DMS row data for the rest of the row)
  5. Commit log record
- If you insert a row into T1 and then roll the operation back, the sequence of log records is as follows:
  1. Start of the out of row data log record
  2. Add LOB data log record (for column C3)
  3. Add LOB data log record (for column C4)
  4. DMS insert record log record (the DMS row data for the rest of the row)
  5. Undo insert log record
  6. Undo start of out of row data log record

**Note:** There are no undo add LOB data log records

7. Abort log record

## LOB manager log records

---

## Appendix C. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics (Task, concept and reference topics)
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF DVD)
  - printed books
- Command-line help
  - Command help
  - Message help

**Note:** The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy 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](http://ibm.com).

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at [ibm.com](http://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 to improve the DB2 documentation, send an email to [db2docs@ca.ibm.com](mailto: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 email 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.

## 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/e-business/linkweb/publications/servlet/pbi.wss](http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss). English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from [www.ibm.com/support/docview.wss?rs=71&uid=swg27009474](http://www.ibm.com/support/docview.wss?rs=71&uid=swg27009474).

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

**Note:** The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

*Table 160. DB2 technical information*

Name	Form Number	Available in print	Availability date
<i>Administrative API Reference</i>	SC27-5506-00	Yes	July 28, 2013
<i>Administrative Routines and Views</i>	SC27-5507-00	No	July 28, 2013
<i>Call Level Interface Guide and Reference Volume 1</i>	SC27-5511-00	Yes	July 28, 2013
<i>Call Level Interface Guide and Reference Volume 2</i>	SC27-5512-00	Yes	July 28, 2013
<i>Command Reference</i>	SC27-5508-00	Yes	July 28, 2013
<i>Database Administration Concepts and Configuration Reference</i>	SC27-4546-00	Yes	July 28, 2013
<i>Data Movement Utilities Guide and Reference</i>	SC27-5528-00	Yes	July 28, 2013
<i>Database Monitoring Guide and Reference</i>	SC27-4547-00	Yes	July 28, 2013
<i>Data Recovery and High Availability Guide and Reference</i>	SC27-5529-00	Yes	July 28, 2013
<i>Database Security Guide</i>	SC27-5530-00	Yes	July 28, 2013
<i>DB2 Workload Management Guide and Reference</i>	SC27-5520-00	Yes	July 28, 2013
<i>Developing ADO.NET and OLE DB Applications</i>	SC27-4549-00	Yes	July 28, 2013
<i>Developing Embedded SQL Applications</i>	SC27-4550-00	Yes	July 28, 2013
<i>Developing Java Applications</i>	SC27-5503-00	Yes	July 28, 2013

Table 160. DB2 technical information (continued)

Name	Form Number	Available in print	Availability date
<i>Developing Perl, PHP, Python, and Ruby on Rails Applications</i>	SC27-5504-00	No	July 28, 2013
<i>Developing RDF Applications for IBM Data Servers</i>	SC27-5505-00	Yes	July 28, 2013
<i>Developing User-defined Routines (SQL and External)</i>	SC27-5501-00	Yes	July 28, 2013
<i>Getting Started with Database Application Development</i>	GI13-2084-00	Yes	July 28, 2013
<i>Getting Started with DB2 Installation and Administration on Linux and Windows</i>	GI13-2085-00	Yes	July 28, 2013
<i>Globalization Guide</i>	SC27-5531-00	Yes	July 28, 2013
<i>Installing DB2 Servers</i>	GC27-5514-00	Yes	July 28, 2013
<i>Installing IBM Data Server Clients</i>	GC27-5515-00	No	July 28, 2013
<i>Message Reference Volume 1</i>	SC27-5523-00	No	July 28, 2013
<i>Message Reference Volume 2</i>	SC27-5524-00	No	July 28, 2013
<i>Net Search Extender Administration and User's Guide</i>	SC27-5526-00	No	July 28, 2013
<i>Partitioning and Clustering Guide</i>	SC27-5532-00	Yes	July 28, 2013
<i>pureXML Guide</i>	SC27-5521-00	Yes	July 28, 2013
<i>Spatial Extender User's Guide and Reference</i>	SC27-5525-00	No	July 28, 2013
<i>SQL Procedural Languages: Application Enablement and Support</i>	SC27-5502-00	Yes	July 28, 2013
<i>SQL Reference Volume 1</i>	SC27-5509-00	Yes	July 28, 2013
<i>SQL Reference Volume 2</i>	SC27-5510-00	Yes	July 28, 2013
<i>Text Search Guide</i>	SC27-5527-00	Yes	July 28, 2013
<i>Troubleshooting and Tuning Database Performance</i>	SC27-4548-00	Yes	July 28, 2013
<i>Upgrading to DB2 Version 10.5</i>	SC27-5513-00	Yes	July 28, 2013
<i>What's New for DB2 Version 10.5</i>	SC27-5519-00	Yes	July 28, 2013
<i>XQuery Reference</i>	SC27-5522-00	No	July 28, 2013

Table 161. DB2 Connect-specific technical information

Name	Form Number	Available in print	Availability date
DB2 Connect Installing and Configuring DB2 Connect Personal Edition	SC27-5516-00	Yes	July 28, 2013
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-5517-00	Yes	July 28, 2013
DB2 Connect User's Guide	SC27-5518-00	Yes	July 28, 2013

---

## Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

### Procedure

To start 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.

---

## Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on [ibm.com](http://ibm.com)<sup>®</sup>.

### About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1>.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is <http://pic.dhe.ibm.com/infocenter/db2luw/v9r8/>.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is <http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/>.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r5>.

---

## Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

**Applicability:** These terms and conditions are in addition to any terms of use for the IBM website.

**Personal use:** You may reproduce these publications for your personal, noncommercial 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.

**Rights:** 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 previous 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.

**IBM Trademarks:** IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)





---

## Appendix D. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about 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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, 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, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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  
U59/3600  
3600 Steeles Avenue East  
Markham, Ontario L3R 9Z7  
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 contains 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample 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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation 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  
    db2DatabaseRestart API 73  
activate database API 367  
activate not logged initially log record 704  
add contact API 34  
add contact group API 36  
add database partition API 373  
add long field record log record 700  
alter column attribute log record 704  
alter table add columns log record 704  
alter table attribute log record 704

### APIs

    change isolation level (REXX) 469  
    changed 19  
    Compress 597  
    db2AddContact 34  
    db2AddContactGroup 36  
    db2AddSnapshotRequest 38  
    db2AdminMsgWrite 40  
    db2ArchiveLog 42  
    db2AutoConfig 44  
    db2AutoConfigFreeMemory 48  
    db2Backup 49  
    db2CfgGet 59  
    db2CfgSet 62  
    db2ConvMonStream 66  
    db2DatabasePing 69  
    db2DatabaseQuiesce 71  
    db2DatabaseRestart 73  
    db2DatabaseUnquiesce 76  
    db2DbDirCloseScan 80  
    db2DbDirGetNextEntry 81  
    db2DbDirOpenScan 85  
    db2DropContact 87  
    db2DropContactGroup 88  
    db2Export 89  
    db2GetAlertCfg 96  
    db2GetAlertCfgFree 101  
    db2GetContactGroup 102  
    db2GetContactGroups 104  
    db2GetContacts 106  
    db2GetDistMap 108  
    db2GetHealthNotificationList 110  
    db2GetRecommendations 112  
    db2GetRecommendationsFree 115  
    db2GetSnapshot 116  
    db2GetSnapshotSize 120  
    db2GetSyncSession 123  
    db2HADRStart 124  
    db2HADRStop 126  
    db2HADRTakeover 128  
    db2HistoryCloseScan 131  
    db2HistoryGetEntry 132  
    db2HistoryOpenScan 134  
    db2HistoryUpdate 138  
    db2Import 141  
    db2Ingest 154  
    db2Inspect 166  
    db2InstanceQuiesce 173  
    db2InstanceStart 176

### APIs (continued)

    db2InstanceStop 182  
    db2InstanceUnquiesce 186  
    db2LdapCatalogDatabase 188  
    db2LdapCatalogNode 190  
    db2LdapDeregister 191  
    db2LdapRegister 192  
    db2LdapUncatalogDatabase 196  
    db2LdapUncatalogNode 197  
    db2LdapUpdate 198  
    db2LdapUpdateAlternateServerForDB 201  
    db2Load 203  
    db2LoadQuery 224  
    db2MonitorSwitches 231  
    db2Prune 234  
    db2QuerySatelliteProgress 237  
    db2ReadLog 239  
    db2ReadLogNoConn 245  
    db2ReadLogNoConnInit 248  
    db2ReadLogNoConnTerm 250  
    db2Recover 251  
    db2Reorg 257  
    db2ResetAlertCfg 267  
    db2ResetMonitor 269  
    db2Restore 272  
    db2Rollforward 286  
    db2Runstats 296  
    db2SelectDB2Copy 307  
    db2SetSyncSession 309  
    db2SetWriteForDB 310  
    db2SpmListIndTrans 312  
    db2SyncSatellite 315  
    db2SyncSatelliteStop 316  
    db2SyncSatelliteTest 317  
    db2UpdateAlertCfg 318  
    db2UpdateAlternateServerForDB 324  
    db2UpdateContact 326  
    db2UpdateContactGroup 328  
    db2UpdateHealthNotificationList 330  
    db2UtilityControl 332  
    db2VendorGetNextObj 566  
    db2VendorQueryApiVersion 568  
    db2XaGetInfo 473  
    db2XaListIndTrans 474  
    Decompress 599  
    GetMaxCompressedSize 600  
    GetSavedBlock 601  
    group plug-in  
        db2secDoesGroupExist 524  
        db2secFreeErrormsg 525  
        db2secFreeGroupListMemory 526  
        db2secGetGroupsForUser 527  
        db2secGroupPluginInit 530  
        db2secPluginTerm 532  
    group retrieval plug-in 522  
    heuristic 471  
    InitCompression 602  
    InitDecompression 603  
    precompiler customization 686  
    REXX syntax 467

## APIs (continued)

- security plug-in
  - db2secClientAuthPluginInit 539
  - db2secClientAuthPluginTerm 541
  - db2secDoesAuthIDExist 542
  - db2secDoesGroupExist 524
  - db2secFreeErrorMsg 525
  - db2secFreeGroupListMemory 526
  - db2secFreeInitInfo 543
  - db2secFreeToken 544
  - db2secGenerateInitialCred 545
  - db2secGetAuthIDs 547
  - db2secGetDefaultLoginContext 549
  - db2secGetGroupsForUser 527
  - db2secGroupPluginInit 530
  - db2secPluginTerm 532
  - db2secProcessServerPrincipalName 551
  - db2secRemapUserid 552
  - db2secServerAuthPluginInit 554
  - db2secServerAuthPluginTerm 557
  - db2secValidatePassword 558
- overview 520
- sqlabndx 334
- sqlaintp 337
- sqlaprep 339
- sqlarbnd 342
- sqlbctcq 345
- sqlbctsq 346
- sqlbftcq 347
- sqlbftpq 349
- sqlbgtss 351
- sqlbmtsq 353
- sqlbotcq 355
- sqlbotsq 357
- sqlbstpq 359
- sqlbstsc 361
- sqlbtcq 364
- sqlcspqy 366
- sqle\_activate\_db 367
- sqle\_deactivate\_db 370
- sqleadn 373
- sqleatcp 375
- sqleatin 378
- sqleAttachToCtx 484
- sqleBeginCtx 485
- sqlecadb 381
- sqlecran 387
- sqlecrea 389
- sqlectnd 397
- sqledcgd 400
- sqledcls 80
- sqleDetachFromCtx 486
- sqledgne 81
- sqledosd 85
- sqledpan 402
- sqledrpd 404
- sqledrpn 406
- sqledtin 408
- sqleEndCtx 487
- sqlefmem 409
- sqlefrce 410
- sqlegdad 413
- sqlegdcl 415
- sqlegdel 416
- sqlegdge 418
- sqlegdgt 420
- sqlegdsc 422

## APIs (continued)

- sqleGetCurrentCtx 489
- sqlegins 423
- sqleInterruptCtx 490
- sqleintr 424
- sqleisig 426
- sqlemgdb 427
- sqlencls 429
- sqlengne 430
- sqlenops 432
- sqleqryc 434
- sqleqryi 436
- sqlesact 438
- sqlesdeg 439
- sqlesetc 441
- sqleseti
  - details 444
- sqleSetTypeCtx 491
- sqleuncd 446
- sqleuncn 448
- sqlgaddr 450
- sqlgdref 451
- sqlgmcpy 452
- sqllogstt 453
- sqludrtd 455
- sqluexpr 89
- sqlugrpn 459
- sqlugtpi 462
- sqluimpr 141, 154
- sqluvdel 569
- sqluvend 570
- sqluvget 572
- sqluvint 574
- sqluvput 578
- sqluvqdp 464
- sqlxhfrg 479
- sqlxphcm 480
- sqlxphrl 481
- summary 1
- TermCompression 604
- TermDecompression 605
- user ID/password plug-ins 533
- application design
  - collating sequence setting 389
  - pointer manipulation
    - copying data 452
    - dereferencing address 451
    - getting variable address 450
  - signal handler installation 426
- applications
  - binding 334
- archive active log API 42
- asynchronous read log API 239
- attach and change password API 375
- attach API 378
- attach to context API 484
- authentication
  - GSS-API 510
  - ID and password 510
  - Kerberos 510
- plug-ins
  - API for checking whether authentication ID exists 542
  - API for cleaning up client authentication plug-in resources 541
  - API for cleaning up resources held by db2secGenerateInitialCred API 543

- authentication (*continued*)
  - plug-ins (*continued*)
    - API for cleaning up server authentication plug-in resources 557
    - API for getting authentication IDs 547
    - API for initializing client authentication plug-in 539
    - API for initializing server authentication plug-in 554
    - API for validating passwords 558
    - APIs for user ID/ password authentication
      - plug-ins 533
    - deploying 493, 494, 496
    - library locations 515
    - security 510
  - security plug-ins 510
  - two-part user IDs 517
- autoconfigure API 44

## B

- backout free log record 693
- backup database API 49
- backups
  - end log record 701
  - vendor products 493, 565
- binarynumerics file type modifier
  - db2load API 203
- bind API
  - sqlabndx 334
- binding
  - application programs to databases 334
  - defaults 334
  - errors 389

## C

- C/C++ language
  - applications
    - include files 30
- catalog database API 381
- catalog database LDAP entry API 188
- catalog DCS database API 413
- catalog node API 397
- catalog node LDAP entry API 190
- change isolation level REXX API 469
- chardel file type modifier
  - db2Export API 89
  - db2Import API 141
  - db2Load API 203
- close table space container query API 345
- close table space query API 346
- COBOL language
  - applications
    - include files 30
    - pointer manipulation 450, 451, 452
- code page file type modifier 203
- code pages
  - db2Export API 89
  - db2Import API 141
  - db2Ingest API 154
- codel file type modifier
  - db2Export API 89
  - db2Import API 141
  - db2Load API 203
- collating sequences
  - user-defined 389

- columns
  - importing 141, 154
- command line processor (CLP)
  - calling APIs from REXX applications 467
- comments
  - database 400
- commit an indoubt transaction API 480
- compound file type modifier
  - db2Import API 141
- COMPR\_CB structure 592
- COMPR\_DB2INFO structure 593
- COMPR\_PIINFO structure 595
- compress a block of data API 597
- compression
  - plug-in interface 493, 590
  - stop the compression library API 604
  - stop the decompression library API 605
- concurrency
  - change isolation level API 469
- convert monitor stream API 66
- copy memory API 452
- create and attach to an application context API 485
- create database API 389
- create database at node API 387
- create index log record 704
- create table log record 704

## D

- data manager log records 704
- data movement
  - between databases 141, 154
- data structures
  - COMPR\_CB 592
  - COMPR\_DB2INFO 593
  - COMPR\_PIINFO 595
  - data 588
  - DB2\_info 580
  - db2DistMapStruct 608
  - db2HistData 609
  - db2LSN 614
  - Init\_output 587
  - Return\_code 589
  - sql\_dir\_entry 615
  - SQLB\_TBS\_STATS 616
  - SQLB\_TBSCONTQRY\_DATA 617
  - SQLB\_TBSPQRY\_DATA 619
  - SQLCA 623
  - sqlchar 624
  - SQLDA 625
  - sqldcol 627
  - sqle\_addn\_options 630
  - sqle\_client\_info 632
  - sqle\_conn\_setting 634
  - sqle\_node\_local 637
  - sqle\_node\_npipe 638
  - sqle\_node\_struct 639
  - sqle\_node\_tcpip 640
  - sqledbdesc 641
  - sqledbterritoryinfo 655
  - sqleninfo 656
  - sqlfupd 659
  - sqllob 668
  - sqlma 669
  - sqlopt 672
  - SQLU\_LSN 674
  - sqlu\_media\_list 675

- data structures (*continued*)
  - SQLU\_RLOG\_INFO 680
  - sqlupi 681
  - SQLXA\_XID 683
  - vendor APIs 565
  - Vendor\_info 583
- database configuration file
  - valid entries 659
- Database Connection Services (DCS) directory
  - adding entries 413
  - cataloging entries 413
  - copying entries 420
  - removing entries 416
  - retrieving entries 418
  - starting scan 422
- database directories
  - retrieving next entry 81
- database quiesce API 71
- databases
  - binding application programs 334
  - concurrent request processing 469
  - creating
    - sqlecrea API 389
  - deleting
    - sqledrpd API 404
  - dropping
    - sqledrpd API 404
  - exporting from table into file
    - db2Export API 89
  - importing from file into table
    - db2Import API 141
    - db2Ingest API 154
  - importing from pipe into table
    - db2Ingest API 154
  - isolating data 469
  - managing 493
- dateformat file type modifier
  - db2Import API 141
  - db2Load API 203
- DB2 Connect
  - connections supported 413
- DB2 Information Center
  - versions 736
- DB2\_info structure 580
- db2AddContact API 34
- db2AddContactGroup API 36
- db2AdminMsgWrite API 40
- db2ArchiveLog API 42
- db2AutoConfig API 44
- db2AutoConfigFreeMemory API 48
- db2Backup API
  - details 49
- db2CfgGet API 59
- db2CfgSet API 62
- db2ConvMonStream API 66
- db2DatabasePing API 69
- db2DatabaseQuiesce API 71
- db2DatabaseRestart API 73
- db2DatabaseUnquiesce API 76
- db2DatabaseUpgrade API
  - details 78
- db2DbDirCloseScan API 80
- db2DbDirGetNextEntry API 81
- db2DbDirOpenScan API 85
- db2DistMapStruct structure 608
- db2DropContact API 87
- db2DropContactGroup API 88
- db2GetAlertCfg API 96
- db2GetAlertCfgFree API 101
- db2GetContactGroup API 102
- db2GetContactGroups API 104
- db2GetContacts API 106
- db2GetDistMap API 108
- db2GetHealthNotificationList API 110
- db2GetRecommendations API 112
- db2GetRecommendationsFree API 115
- db2GetSnapshot API
  - details 116
  - estimating output buffer size 120
- db2GetSnapshotSize API 120
- db2GetSyncSession API 123
- db2HADRStart API 124
- db2HADRStop API 126
- db2HADRTakeover API 128
- db2HistData structure 609
- db2HistoryCloseScan API 131
- db2HistoryGetEntry API 132
- db2HistoryOpenScan API 134
- db2HistoryUpdate API 138
- DB2Ingest API 154
- db2Inspect API 166
- db2InstanceQuiesce API 173
- db2InstanceStart API 176
- db2InstanceStop API 182
- db2InstanceUnquiesce API 186
- db2LdapCatalogDatabase API 188
- db2LdapCatalogNode API 190
- db2LdapDeregister API 191
- db2LdapRegister API 192
- db2LdapUncatalogDatabase API 196
- db2LdapUncatalogNode API 197
- db2LdapUpdate API 198
- db2LdapUpdateAlternateServerForDB API 201
- db2Load API 203
- db2LoadQuery API 224
- db2LSN data type 23
- db2LSN structure 614
- db2MonitorSwitches API 231
- db2Prune API 234
- db2QuerySatelliteProgress API 237
- db2ReadLog API 239
- db2ReadLogNoConn API 245
- db2ReadLogNoConnInit API 248
- db2ReadLogNoConnTerm API 250
- db2Recover API
  - details 251
- db2Reorg API 257
- db2ResetAlertCfg API 267
- db2ResetMonitor API 269
- db2Restore API
  - details 272
- db2Rollforward API
  - details 286
- db2Runstats API 296
- db2SelectDB2Copy API
  - details 307
- db2SetSyncSession API 309
- db2SetWriteForDB API 310
- db2SyncSatellite API 315
- db2SyncSatelliteStop API 316
- db2SyncSatelliteTest API 317
- db2UpdateAlertCfg API 318
- db2UpdateAlternateServerForDB API 324
- db2UpdateContact API 326



- db2UpdateContactGroup API 328
- db2UpdateHealthNotificationList API 330
- db2UtilityControl API 332
- db2VendorGetNextObj API 566
- db2VendorQueryApiVersion API 568
- db2XaGetInfo API 473
- db2XaListIndTrans API 474
- DDL statement log record 722
- deactivate database API 370
- debugging
  - security plug-ins 519
- decompress a block of data API 599
- delete alias for node from LDAP server API 197
- delete committed session API 569
- delete long field record log record 700
- delete record log record 704
- delete record to empty page log record 704
- dereference address API 451
- deregister database from LDAP server API 196
- detach from and free memory associated with application
  - context API 487
- detach from context API 486
- detach from instance API 408
- directories
  - Database Connection Services (DCS)
    - adding entries 413
    - copying entries 420
    - deleting entries 416
    - retrieving entries 418
    - starting scan 422
  - local database
    - retrieving entries 81
    - starting scan 85
  - node
    - adding entries 397
    - deleting entries 448
    - details 448
    - retrieving entries 430
  - system database
    - adding entries 381
    - cataloging databases 381
    - deleting entries 446
    - retrieving entries 81
    - starting scan 85
    - uncataloging databases 446
- documentation
  - overview 733
  - PDF files 734
  - printed 734
  - terms and conditions of use 736
- drop contact API 87
- drop contact group API 88
- drop database API 404
- drop database on database partition server API 402
- drop index log record 704
- drop table log record 704

## E

- end database directory scan API 80
- end database history records scan API 131
- end DCS directory scan API 415
- end node directory scan API 429
- error messages
  - binding 334
  - retrieving
    - sqlaintp API 337

- error messages (*continued*)
  - security plug-ins 507
- errors
  - rollforward 286
- export API 89
- exports
  - data
    - db2Export API 89
    - file type modifiers 89
- extended database description block 649

## F

- fastparse file type modifier
  - db2Load API 203
- fetch query data for rows in table space API 349
- fetch query data for rows in table space container API 347
- file type modifiers
  - export API 89
  - import API 141
  - load API 203
- force application API 410
- forcein file type modifier
  - db2Import API 141
  - db2Load API 203
- forget transaction status API 479
- formatted user data record log record 704
- FORTTRAN language
  - applications
    - include files 30
  - pointer manipulation 450, 451, 452
- free autoconfigure memory API 48
- free db2GetAlertCfg memory API 101
- free db2GetRecommendations memory API 115
- free memory allocated by sqlbctq and sqlbmtsq APIs API 409

## G

- generatedignore file type modifier
  - db2Import API 141
  - db2load API 203
- generatedmissing file type modifier
  - db2Import API 141
  - db2load API 203
- generatedoverride file type modifier
  - db2load API 203
- get address API 450
- get alert configuration API 96
- get configuration parameters API 59
- get contact group API 102
- get contact groups API 104
- get contacts API 106
- get current context API 489
- get DCS directory entries API 420
- get DCS directory entry for database API 418
- get error message API
  - details 337
- get health notification list API 110
- get information for resource manager API 473
- get instance API 423
- get next database directory entry API 81
- get next entry in database history records API 132
- get next node directory entry API 430
- get or update monitor switches API 231
- get recommendations for health indicator in alert state
  - API 112

- get row distribution number API 459
- get satellite sync session API 123
- get snapshot API 116
- get SQLSTATE message API 453
- get table space statistics API 351
- global pending list log record 693
- GSS-APIs
  - authentication plug-ins 561

## H

- help
  - SQL statements 736
- heuristic abort log record 693
- heuristic commit log record 693
- highlighting conventions viii
- host systems
  - connections supported by DB2 Connect
    - sqlegdad API 413
- how this book is structured vii

## I

- identityignore file type modifier
  - db2Import API 141
  - db2Load API 203
- identitymissing file type modifier
  - db2Import API 141
  - db2Load API 203
- identityoverride file type modifier
  - db2Load API 203
- implieddecimal file type modifier
  - db2Import API 141
  - db2Load API 203
- import API 141
- import replace (truncate) log record 704
- imports
  - code page considerations 141, 154
  - database access through DB2 Connect 141, 154
  - file to database table 141, 154
  - file-type modifiers 141, 154
  - PC/IXF files
    - multiple-part 141, 154
  - restrictions 141, 154
  - to hierarchy that does not exist 141, 154
  - to remote database 141, 154
  - to table that does not exist 141, 154
  - to typed tables 141, 154
- include files
  - overview 30
- indexfreespace file type modifier
  - db2Import API 203
- indexixf file type modifier
  - db2Import API 141
- indexschema file type modifier
  - db2Import API 141
- indoubt transactions
  - rolling back using API 481
- Ingest API 154
- Init\_output structure 587
- INIT-INPUT structure 585
- initialize and link to device API 574
- initialize compression library API 602
- initialize decompression library API 603
- initialize read log without database connection API 248
- initialize table log record 704

- insert record log record 704
- insert record to empty page log record 704
- inspect database API 166
- install signal handler API 426
- instance quiesce API 173
- instance start API 176
- instance stop API 182
- instance unquiesce API 186
- interrupt API 424
- interrupt context API 490
- isolation levels
  - changing 469

## K

- keepblanks file type modifier
  - db2Import API 141
  - db2Load API 203
- Kerberos authentication protocol
  - plug-ins
    - deploying 496
  - samples 564

## L

- LDAP
  - deregister server API 191
  - register server API 192
  - update alternate server for database API 201
  - update server API 198
- libraries
  - security plug-ins
    - loading in DB2 498
    - restrictions 500
- list DRDA indoubt transactions API 366
- list indoubt transactions API 474
- list SPM indoubt transactions API
  - details 312
- load API 203
- load pending list log record 701
- load query API 224
- load start log record
  - details 701
- load utility
  - file type modifiers 203
- LOB manager log records 728
- lobsinfile file type modifier
  - db2Export API 89
  - db2Import API 141
  - db2Load API 203
- local database directory
  - retrieving entries 81
  - starting scan 85
- local pending list log record 693
- locks
  - changing 469
- log records
  - activate not logged initially 704
  - backout free 693
  - backup end 701
  - changing 704
  - creating 704
  - data manager 704
  - DDL statement log record 722
  - delete 704

- log records (*continued*)
  - deleting
    - data manager 704
    - long field 700
  - dropping
    - indexes 704
    - tables 704
  - global pending list 693
  - headers 691
  - heuristic abort 693
  - heuristic commit 693
  - import replace (truncate) 704
  - initialize tables 704
  - insert records 704
  - load delete start compensation 701
  - load pending list 701
  - load start 701
  - LOB manager 728
  - local pending list 693
  - log manager header 691
  - long field manager 700
  - MPP coordinator commit 693
  - MPP subordinator commit 693
  - MPP subordinator prepare 693
  - non-update long field 700
  - normal abort 693
  - normal commit 693
  - partition information log record 722
  - rename schema 704
  - rename table 704
  - reorg table 704
  - rollback add columns 704
  - rollback create table 704
  - rollback delete record 704
  - rollback drop table 704
  - rollback insert 704
  - rollback partition state record 704
  - rollback update record 704
  - summary 688
  - system catalog migration end 701
  - system catalog migration start 701
  - table details 704
  - table load delete start 701
  - table space roll forward to point in time ends 701
  - table space roll forward to point in time starts 701
  - table space rolled forward 701
  - TM prepare 693
  - transaction manager 693
  - undo change columns 704
  - undo DDL statement 722
  - undo partition state 704
  - undo rename schema 704
  - undo rename table 704
  - update partition state 704
  - update records 704
  - utility manager 701
  - XA prepare 693
- log sequence numbers (LSNs)
  - changes 23
  - headers 691
  - log manager log record headers 691
- logs
  - recovery 389
- long field manager log records 700

## M

- migration
  - databases
    - sqlcmgdb API 427
  - log records 701
- MPP coordinator commit log record 693
- MPP subordinator commit log record 693
- MPP subordinator prepare log record 693
- multiple concurrent requests
  - changing isolation levels 469

## N

- nochecklengths file type modifier
  - db2Import API 141
  - db2Load API 203
- node directories
  - adding entries 397
  - deleting entries 448
  - retrieving entries 430
- nodefaults file type modifier
  - db2Import API 141
- nodes
  - SOCKS
    - sqle\_node\_tcpip data structure 640
  - start DCS directory scan API 422
- nodoubledel file type modifier
  - db2Export API 89
  - db2Import API 141
  - db2Load API 203
- noeofchar file type modifier
  - db2Import API 141
  - db2Load API 203
- noheader file type modifier
  - db2Load API 203
- non-update long field record log record 700
- normal abort log record 693
- normal commit log record 693
- norowwarnings file type modifier
  - db2Load API 203
- notices 739
- notypeid file type modifier
  - db2Import API 141
- nullindchar file type modifier
  - db2Import API 141
  - db2Load API 203

## O

- open table space container query API 355
- open table space query API 357

## P

- packages
  - creating
    - sqlabndx API 334
  - re-creating 342
- partition information log record 722
- partitioned database environments
  - table distribution information 108, 462
- passwords
  - changing
    - sqleatcp API 375

- performance
  - tables
    - reorganizing 257
- ping database API 69
- plug-ins
  - database management 493
  - group retrieval 522
  - GSS-API authentication 561
  - security
    - APIs 508, 520
    - deploying 493, 494, 495, 496
    - error messages 507
    - naming conventions 516
    - restrictions (GSS-API authentication) 563
    - restrictions (plug-in libraries) 500
    - restrictions (summary) 502
    - return codes 504
    - samples 564
    - versions 518
  - user ID/password authentication 533
- pointer manipulation
  - copying data between memory areas 452
  - dereferencing addresses 451
  - getting addresses of variables 450
- precompile application program API 339
- privileges
  - databases
    - granted when creating database 389
- problem determination
  - security plug-ins 519
- prune history file API 234

## Q

- query client API 434
- query client information API 436
- query satellite synchronization session API 237
- quiesce table spaces for table API 464

## R

- read log without a database connection API 245
- reading data from device API 572
- rebind API 342
- reclen file type modifier
  - db2Import API 141
  - db2Load API 203
- record identifiers (RIDs)
  - log records 691
- recover database API 251
- redistribute data across database partition group API 455
- redistribution of data
  - database partition groups 455
- rename schema log record 704
- rename table log record 704
- reorg table log record 704
- reorganize API 257
- reset alert configuration API 267
- reset database system monitor data API 269
- restart database API 73
- restore database API 272
- restores
  - vendor products 493, 565
- return codes
  - APIs 27
- Return\_code structure 589

- REXX language
  - APIs
    - calling DB2 CLP 467
    - syntax 467
- rollback add columns log record 704
- rollback create table log record 704
- rollback delete record log record 704
- rollback delete record to empty page log record 704
- rollback drop table log record 704
- rollback insert log record 704
- rollback insert record to empty page log record 704
- rollback update record log record 704
- rollforward recovery
  - db2Rollforward API 286
- runstats API 296

## S

- satellite synchronization API 315
- schemas
  - sqlecrea API 389
- security
  - plug-ins
    - 32-bit considerations 519
    - 64-bit considerations 519
    - API calling sequence 508
    - APIs 520, 524, 525, 526, 527, 530, 532, 539, 541, 542, 543, 544, 545, 547, 549, 551, 552, 554, 557, 558
    - APIs (group retrieval) 522
    - APIs (GSS-API) 561
    - APIs (user ID/password) 533
    - APIs (versions) 518
    - deploying 493, 494, 495, 496, 502, 510
    - developing 500, 510
    - enabling 510
    - error messages 507
    - group retrieval 493
    - GSS-API (deploying) 495
    - GSS-API (restrictions) 563
    - initialization 498
    - Kerberos 496
    - libraries 515
    - loading 498, 510
    - naming 516
    - overview 510
    - problem determination 519
    - restrictions on libraries 500
    - return codes 504
    - SQLCODE values 519
    - SQLSTATE values 519
    - two-part user ID support 517
    - user ID/password 494
  - samples 564
    - user ID and password 493
- set accounting string API 438
- set application context type API 491
- set client API 441
- set client information API 444
- set configuration parameters API 62
- set maximum degree of intrapartition parallelism API 439
- set satellite sync session API 309
- set table space containers API 361
- signal handlers
  - install signal handler API 426
  - interrupt application requests API 424
- single table space query API 359

- snapshot monitoring
  - adding requests 38
- SOCKS
  - nodes
    - sqle\_node\_tcpip data structure 640
- SQL statements
  - help
    - displaying 736
- sql\_dir\_entry structure 615
- sqlabndx API 334
- sqlaintp API 337
- sqlaprep API 339
- sqlarbind API 342
- SQLB\_TBS\_STATS structure 616
- SQLB\_TBSCONTQRY\_DATA structure 617
- SQLB\_TBSPQRY\_DATA structure 619
- sqlbctcq API 345
- sqlbctsq API 346
- sqlbftcq API 347
- sqlbftpq API 349
- sqlbgtss API 351
- sqlbmtsq API 353
- sqlbotcq API 355
- sqlbotsq API 357
- sqlbstpq API 359
- sqlbstsc API 361
- sqlbtcq API 364
- SQLCA structure
  - details 623
  - retrieving error messages 27, 337, 453
- sqlchar structure
  - details 624
- SQLCODE
  - APIs 27
- sqlcspqy API 366
- SQLDA
  - overview 625
- SQLDB2 API
  - calling DB2 APIs in REXX 467
- sqldcol structure 627
- sqle\_activate\_db API 367
- sqle\_addn\_options structure 630
- sqle\_client\_info structure 632
- sqle\_conn\_setting structure 634
- sqle\_deactivate\_db API 370
- sqle\_node\_local structure 637
- sqle\_node\_npipe structure 638
- sqle\_node\_struct structure 639
- sqle\_node\_tcpip structure 640
- sqleaddn API 373
- sqleatcp API 375
- sqleatin API 378
- sqleAttachToCtx API
  - details 484
- sqleBeginCtx API
  - details 485
- sqlecadb API 381
- sqlecran API 387
- sqlecrea API 389
- sqlectnd API 397
- sqledbdesc structure 641
- sqledbdescext structure 649
- sqledbterritoryinfo structure 655
- sqledcgd API 400
- sqledcls API 80
- sqleDetachFromCtx API
  - details 486
- sqledgne API 81
- sqledosd API 85
- sqledpan API 402
- sqledrpd API 404
- sqledrpn API 406
- sqledtin API 408
- sqleEndCtx API
  - details 487
- sqlefmem API 409
- sqlefrce API 410
- sqlegdad API 413
- sqlegdcl API 415
- sqlegdel API 416
- sqlegdge API 418
- sqlegdgt API 420
- sqlegdsc API 422
- sqleGetCurrentCtx API
  - details 489
- sqlegins API 423
- sqleInterruptCtx API
  - details 490
- sqleintr API 424
- sqleisig API 426
- sqlemgdb API 427
- sqlencls API 429
- sqlengne API 430
- sqleninfo structure 656
- sqlenops API 432
- sqleqryc API 434
- sqleqryi API 436
- sqlsact API 438
- sqlsdeg API 439
- sqlsetc API 441
- sqlseti API
  - details 444
- sqleSetTypeCtx API
  - details 491
- sqleuncd API 446
- sqleuncn API 448
- sqlfupd structure 659
- sqlgaddr API 450
- sqlgdref API 451
- sqlgmcpy API 452
- sqllob structure 668
- sqlma structure 669
- sqllogstt API 453
- sqlopt structure 672
- SQLSTATE
  - APIs 27
  - retrieving messages 453
- SQLU\_LSN structure 674
- sqlu\_media\_list structure 675
- SQLU\_RLOG\_INFO structure 680
- sqludrdr API 455
- sqluexpr API 89
- sqlugrpn API 459
- sqlugtpi API 462
- sqluimpr API 141, 154
- sqlupi structure 681
- sqluvdel API 569
- sqluvend API 570
- sqluvget API 572
- sqluvint API 574
- sqluvput API 578
- sqluvqdp API 464
- SQLWARN
  - APIs 27

- SQLXA\_XID structure 683
- sqlxhfrg API 479
- sqlxphcm API 480
- sqlxphrl API 481
- start database directory scan API 85
- start database history records scan API 134
- start DCS directory scan API 422
- start HADR API 124
- start node directory scan API 432
- stop HADR API 126
- stop satellite sync API 316
- system database directory
  - adding entries 381
  - cataloging databases 381
  - deleting entries 446
  - retrieving entries 81
  - starting scan 85
  - uncataloging databases 446

## T

- table space container query API 364
- table space query API 353
- table spaces
  - recovering
    - rollforward recovery 701
  - rollforward recovery
    - log records 701
- tables
  - exporting data to files 89
  - importing data 141, 154
  - loading 701
- take over as primary database API 128
- TCP/IP
  - SOCKS 640
- terminate reading logs without database connection API 250
- termination
  - abnormal 73
- terms and conditions
  - publications 736
- test satellite synchronization API 317
- third party plug-ins 493
- timeformat file type modifier
  - db2Import API 141
  - db2Load API 203
- timestampformat file type modifier
  - db2import API 141
  - db2load API 203
- TM prepare log record 693
- totalreespace file type modifier
  - db2Load API 203
- transaction identifier log records 691
- transaction managers
  - log records 693
- troubleshooting
  - security plug-ins 519

## U

- uncatalog database API 446
- uncatalog database from DCS directory API 416
- uncatalog entry from node directory API 448
- uncataloging
  - system database directory 446
- undo alter column attribute log record 704
- undo DDL statement log record 722

- undo rename schema log record 704
- undo rename table log record 704
- unlink device and release its resources API 570
- unquiesce database API 76
- update alert configuration API 318
- update alternate server for database API 324
- update contact API 326
- update contact group API 328
- update database history records API 138
- update health notification list API 330
- update record log record 704
- upgrade database API 78
- usedefaults file type modifier
  - db2Import API 141
  - db2Load API 203
- user IDs
  - two-part 517
- utility control API 332
- utility log records
  - summary 701

## V

- vendor database plug-ins 493
- vendor products
  - APIs
    - backups 565
    - restores 565
  - Data structure 588
  - Init\_input structure 585
  - Vendor\_info structure 583

## W

- who should use this book vii
- write data to a vendor device API 578

## X

- XA prepare log record 693

## Z

- zoned decimal file type modifier
  - db2Load API 203





Printed in USA

SC27-5506-00





Spine information:

IBM DB2 10.5 for Linux, UNIX, and Windows

Administrative API Reference

