IBM DB2 Universal Database

API Reference

Version 5

S10J-8167-00

IBM DB2 Universal Database

API Reference

Version 5

S10J-8167-00

Before using this information and the product it supports, be sure to read the general information under Appendix I, "Notices" on page 539.

Contents

About This Book

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

- C
- 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, COBOL, FORTRAN, or REXX programming language
- Application program design.

How this Book is Structured

This book provides the reference information needed to develop administrative applications.

The following topics are covered:

Chapter 1	Provides a description of all database manager APIs.
Chapter 2	Describes DB2 APIs that are only supported in the REXX programming language.
Chapter 3	Describes data structures used when calling APIs.
Appendix A	Explains the conventions used to name objects such as databases and tables.
Appendix B	Provides a description of transaction and heuristic APIs.
Appendix C	Describes how to contact IBM for information about the function and use of APIs that enable the customization of precompilers.
Appendix D	Describes the function and use of APIs that enable DB2 to interface with other vendor software.
Appendix E	Describes new APIs that permit the allocation of separate environments or contexts for each thread within a process, enabling true concurrent access to a DB2 database.

Appendix F	Provides information on extracting and working with DB2 log records.
Appendix G	Discusses issues that should be considered before migrating an application to DB2 Version 2.

Chapter 1. Application Programming Interfaces

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

Note: Slashes (/) in directory paths are specific to UNIX based systems, and are equivalent to back slashes (\) in directory paths on OS/2 and Windows operating systems.

DB2 APIs

The following table lists the APIs grouped by functional category:

Table 1 (Page 1 of 5). DB2 APIs

API Description	API Function Name [b]	Sample Code [c d]	INCLUDE File [e f]
Database Manager Control			
START DATABASE MANAGER	sqlepstart	makeapi	sqlenv
STOP DATABASE MANAGER	sqlepstp	makeapi, dbstop	sqlenv
GET DATABASE MANAGER CONFIGURATION	sqlfxsys	dbmconf	sqlutil
GET DATABASE MANAGER CONFIGURATION DEFAULTS	sqlfdsys	d_dbmcon	sqlutil
RESET DATABASE MANAGER CONFIGURATION	sqlfrsys	dbmconf	sqlutil
UPDATE DATABASE MANAGER CONFIGURATION	sqlfusys	dbmconf	sqlutil
SET RUNTIME DEGREE	sqlesdeg	setrundg	sqlenv
Database Control			
RESTART DATABASE	sqlerstd	restart	sqlenv
CREATE DATABASE	sqlecrea	dbconf	sqlenv
CREATE DATABASE AT NODE	sqlecran	n/a	sqlenv
DROP DATABASE	sqledrpd	dbconf	sqlenv
DROP DATABASE AT NODE	sqledpan	n/a	sqlenv
MIGRATE DATABASE	sqlemgdb	migrate	sqlenv
LIST INDOUBT TRANSACTIONS	sqlxphqr	n/a	sqlxa
ACTIVATE DATABASE	sqle_acti-vate_db	n/a	sqlenv
DEACTIVATE DATABASE	sqle_deac-tivate_db	n/a	sqlenv

Table 1 (Page 2 of 5). DB2 APIs

API Description	API Function Name [b]	Sample Code [c d]	INCLUDE File [e f]
LIST DRDA INDOUBT TRANSACTIONS	sqlcspqy	n/a	sqlxa
Database Directory Management			
CATALOG DATABASE	sqlecadb	dbcat	sqlenv
UNCATALOG DATABASE	sqleuncd	dbcat	sqlenv
CATALOG DCS DATABASE	sqlegdad	dcscat	sqlenv
UNCATALOG DCS DATABASE	sqlegdel	dcscat	sqlenv
CHANGE DATABASE COMMENT	sqledcgd	dbcmt	sqlenv
OPEN DATABASE DIRECTORY SCAN	sqledosd	dbcat	sqlenv
GET NEXT DATABASE DIRECTORY ENTRY	sqledgne	dbcat	sqlenv
CLOSE DATABASE DIRECTORY SCAN	sqledcls	dbcat	sqlenv
OPEN DCS DIRECTORY SCAN	sqlegdsc	dcscat	sqlenv
GET DCS DIRECTORY ENTRIES	sqlegdgt	dcscat	sqlenv
CLOSE DCS DIRECTORY SCAN	sqlegdcl	dcscat	sqlenv
GET DCS DIRECTORY ENTRY FOR DATABASE	sqlegdge	dcscat	sqlenv
Client/Server Directory Management			
CATALOG NODE	sqlectnd	nodecat	sqlenv
UNCATALOG NODE	sqleuncn	nodecat	sqlenv
OPEN NODE DIRECTORY SCAN	sqlenops	nodecat	sqlenv
GET NEXT NODE DIRECTORY ENTRY	sqlengne	nodecat	sqlenv
CLOSE NODE DIRECTORY SCAN	sqlencls	nodecat	sqlenv
Network Support			
REGISTER	sqleregs	regder	sqlenv
DEREGISTER	sqledreg	regder	sqlenv
Database Configuration			
GET DATABASE CONFIGURATION	sqlfxdb	dbconf	sqlutil
GET DATABASE CONFIGURATION DEFAULTS	sqlfddb	d_dbconf	sqlutil
RESET DATABASE CONFIGURATION	sqlfrdb	dbconf	sqlutil
UPDATE DATABASE CONFIGURATION	sqlfudb	dbconf	sqlutil
Backup/Recovery			
BACKUP DATABASE	sqlubkp	backrest	sqlutil
RESTORE DATABASE	sqlurst	backrest	sqlutil
ROLLFORWARD DATABASE	sqluroll	backrest	sqlutil

Table 1 (Page 3 of 5). DB2 APIs

API Description	API Function Name [b]	Sample Code [c d]	INCLUDE File [e f]
OPEN RECOVERY HISTORY FILE SCAN	`sqluhops`	`rechist`	`sqlutil`
GET NEXT RECOVERY HISTORY FILE ENTRY	`sqluhgne`	`rechist`	`sqlutil`
CLOSE RECOVERY HISTORY FILE SCAN	`sqluhcls`	`rechist`	`sqlutil`
PRUNE RECOVERY HISTORY FILE	`sqluhprn`	`rechist`	`sqlutil`
UPDATE RECOVERY HISTORY FILE	`sqluhupd`	`rechist`	`sqlutil`
Operational Utilities			
FORCE APPLICATION	`sqlefrce`	`dbstop`	`sqlenv`
REORGANIZE TABLE	`sqlureot`	`dbstat`	`sqlutil`
RUNSTATS	`sqlustat`	`dbstat`	`sqlutil`
Database Monitoring			
ESTIMATE SIZE REQUIRED FOR sqlmonss() OUTPUT BUFFER	`sqlmonsz`	`monsz`	`sqlmon`
GET/UPDATE MONITOR SWITCHES	`sqlmon`	`n/a`	`sqlmon`
GET SNAPSHOT	`sqlmonss`	`dbsnap`	`sqlmon`
RESET MONITOR	`sqlmrset`	`monreset`	`sqlmon`
Data Utilities			
EXPORT	`sqluexpr`	`impexp`	`sqlutil`
IMPORT	`sqluimpr`	`impexp`	`sqlutil`
LOAD	`sqluload`	`tload`	`sqlutil`
LOAD QUERY	`sqluqry`	`qload`	`sqlutil`
General Application Programming			
GET ERROR MESSAGE	`sqlaintp`	`util, checkerr`	`sql`
GET SQLSTATE MESSAGE	`sqlogstt`	`util, checkerr`	`sql`
INSTALL SIGNAL HANDLER	`sqleisig`	`util, checkerr`	`sqlenv`
INTERRUPT	`sqleintr`	`util, checkerr`	`sqlenv`
DEREFERENCE ADDRESS	`sqlgdref`	`nodecat`	`sqlutil`
COPY MEMORY	`sqlgmcpy`	`tspace`	`sqlutil`
FREE MEMORY	`sqlefmem`	`tabspace, tspace`	`sqlenv`
GET ADDRESS	`sqlgaddr`	`dbmconf`	`sqlutil`

Table 1 (Page 4 of 5). DB2 APIs

API Description	API Function Name b	Sample Code c d	INCLUDE File e f
Application Preparation			
PRECOMPILE PROGRAM	sqlaprep	makeapi	sql
BIND	sqlabndx	makeapi	sql
REBIND	sqlarbnd	rebind	sql
Remote Server Utilities			
ATTACH	sqleatin	dbinst	sqlenv
DETACH	sqledtin	dbinst	sqlenv
Table Space Management			
TABLESPACE CONTAINER QUERY	sqlbtcq	tabscont	sqlutil
OPEN TABLESPACE CONTAINER QUERY	sqlbotcq	tabscont	sqlutil
FETCH TABLESPACE CONTAINER QUERY	sqlbftcq	tabscont	sqlutil
CLOSE TABLESPACE CONTAINER QUERY	sqlbctcq	tabscont	sqlutil
SET TABLESPACE CONTAINERS	sqlbstsc	backrest	sqlutil
TABLESPACE QUERY	sqlbmtsq	tabspace	sqlutil
SINGLE TABLESPACE QUERY	sqlbstpq	tabspace	sqlutil
OPEN TABLESPACE QUERY	sqlbotsq	tabspace	sqlutil
FETCH TABLESPACE QUERY	sqlbftpq	tabspace	sqlutil
CLOSE TABLESPACE QUERY	sqlbctsq	tabspace	sqlutil
GET TABLESPACE STATISTICS	sqlbgtss	tabspace	sqlutil
QUIESCE TABLESPACES FOR TABLE	sqluvqdp	tquiesce	sqlutil
Node Management			
ADD NODE	sqleaddn	n/a	sqlenv
DROP NODE VERIFY	sqledrpn	n/a	sqlenv
Nodegroup Management			
REDISTRIBUTE NODEGROUP	sqludrdt	n/a	sqlutil
Additional APIs			
GET AUTHORIZATIONS	sqluadau	dbauth	sqlutil
GET INSTANCE	sqlegins	dbinst	sqlenv
QUERY CLIENT	sqleqryc	client	sqlenv
SET CLIENT	sqlesetc	client	sqlenv
SET ACCOUNTING STRING	sqlesact	setact	sqlenv
ASYNCHRONOUS READ LOG	sqlurlog	n/a	sqlutil
GET ROW PARTITIONING NUMBER	sqlugrpn	n/a	sqlutil

Table 1 (Page 5 of 5). DB2 APIs			
API Description	**API Function Name** [b]	**Sample Code** [c d]	**INCLUDE File** [e f]
GET TABLE PARTITIONING INFORMATION	`sqlugtpi`	`n/a`	`sqlutil`

Note:

[a] This is a pre-version 2 API and is not supported on all platforms.

[b] The fourth character of the generic API function name is always `g`.

[c] The sample programs can be found in the language specific directory of the `samples` directory in the `sqllib` directory (for example, `sqllib\samples\c` for C source code).

[d] The file extensions on sample code depend on the programming language being used. For example, for sample code written in C, the extension is `.c` or `.sqc`. Not all programs are available in all supported programming languages. Not all APIs have sample code (indicated by n/a).

[e] The file extensions on INCLUDE files depend on the programming language being used. For example, an INCLUDE file written for C has a file extension of `.h`.

[f] The INCLUDE files can be found in directory `sqllib\include` (directory delimiters are dependant upon the operating system).

DB2 Sample Programs

The following table lists the APIs grouped by sample program:

Table 2 (Page 1 of 4). DB2 APIs by Sample Program

Sample Code	**Included APIs**
backrest	sqlbstsc - Set Tablespace Containers sqlubkp - Backup Database sqluroll - Rollforward Database sqlurst - Restore Database
checkerr	sqlaintp - Get Error Message sqleintr - Interrupt sqleisig - Install Signal Handler sqlogstt - Get SQLSTATE Message
client	sqleqryc - Query Client sqlesetc - Set Client
d_dbconf	sqlfddb - Get Database Configuration Defaults
d_dbmcon	sqlfdsys - Get Database Manager Configuration Defaults
dbauth	sqluadau - Get Authorizations
dbcat	sqlecadb - Catalog Database sqledcls - Close Database Directory Scan sqledgne - Get Next Database Directory Entry sqledosd - Open Database Directory Scan sqleuncd - Uncatalog Database

Table 2 (Page 2 of 4). DB2 APIs by Sample Program

Sample Code	Included APIs
dbcmt	sqledcgd - Change Database Comment
dbconf	sqlecrea - Create Database sqledrpd - Drop Database sqlfrdb - Reset Database Configuration sqlfudb - Update Database Configuration sqlfxdb - Get Database Configuration
dbinst	sqleatin - Attach sqledtin - Detach sqlegins - Get Instance
dbmconf	sqlfrsys - Reset Database Manager Configuration sqlfusys - Update Database Manager Configuration sqlfxsys - Get Database Manager Configuration sqlgaddr - Get Address
dbsnap	sqlmonss - Get Snapshot
dbstat	sqlureot - Reorganize Table sqlustat - Runstats
dbstop	sqlefrce - Force Application sqlepstp - Stop Database Manager
dcscat	sqlegdad - Catalog DCS Database sqlegdcl - Close DCS Directory Scan sqlegdel - Uncatalog DCS Database sqlegdge - Get DCS Directory Entry for Database sqlegdgt - Get DCS Directory Entries sqlegdsc - Open DCS Directory Scan
impexp	sqluexpr - Export sqluimpr - Import
makeapi	sqlabndx - Bind sqlaprep - Precompile Program sqlepstp - Stop Database Manager sqlepstr - Start Database Manager
migrate	sqlemgdb - Migrate Database
monreset	sqlmrset - Reset Monitor
monsz	sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer
nodecat	sqlectnd - Catalog Node sqlencls - Close Node Directory Scan sqlengne - Get Next Node Directory Entry sqlenops - Open Node Directory Scan sqleuncn - Uncatalog Node sqlgdref - Dereference Address
qload	sqluqry - Load Query
rebind	sqlarbnd - Rebind

Table 2 (Page 3 of 4). DB2 APIs by Sample Program

Sample Code	Included APIs
rechist	sqluhcls - Close Recovery History File Scan sqluhgne - Get Next Recovery History File Entry sqluhops - Open Recovery History File Scan sqluhprn - Prune Recovery History File sqluhupd - Update Recovery History File
regder	sqledreg - Deregister sqleregs - Register
restart	sqlerstd - Restart Database
setact	sqlesact - Set Accounting String
setrundg	sqlesdeg - Set Runtime Degree
tabscont	sqlbctcq - Close Tablespace Container Query sqlbftcq - Fetch Tablespace Container Query sqlbotcq - Open Tablespace Container Query sqlbtcq - Tablespace Container Query
tabspace	sqlbctsq - Close Tablespace Query sqlbftpq - Fetch Tablespace Query sqlbgtss - Get Tablespace Statistics sqlbmtsq - Tablespace Query sqlbotsq - Open Tablespace Query sqlbstpq - Single Tablespace Query sqlefmem - Free Memory
tload	sqluload - Load
tquiesce	sqluvqdp - Quiesce Tablespaces for Table
tspace	sqlefmem - Free Memory sqlgmcpy - Copy Memory
util	sqlaintp - Get Error Message sqleintr - Interrupt sqleisig - Install Signal Handler sqlogstt - Get SQLSTATE Message
n/a	sqlcspqy - List DRDA Indoubt Transactions sqle_activate_db - Activate Database sqle_deactivate_db - Deactivate Database sqleaddn - Add Node sqlecran - Create Database at Node sqledpan - Drop Database at Node sqledrpn - Drop Node Verify sqludrdt - Redistribute Nodegroup sqlugrpn - Get Row Partitioning Number sqlugtpi - Get Table Partitioning Information sqlurlog - Asynchronous Read Log sqlxphqr - List Indoubt Transactions

Table 2 (Page 4 of 4). DB2 APIs by Sample Program

Sample Code	Included APIs
Note: [a] The sample programs can be found in the language specific directory of the `samples` directory in the `sqllib` directory (for example, `sqllib\samples\c` for C source code). The file extensions on sample code depend on the programming language being used. For example, for sample code written in C, the extension is `.c` or `.sqc`. Not all programs are available in all supported programming languages. Not all APIs have sample code (indicated by n/a).	

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-node system, the scope is that single node only. In a multi-node system, it is the collection of all logical nodes defined in the node configuration file, `db2nodes.cfg`.

Authorization

The authority required to successfully call the API.

Required Connection

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

API Include File

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

C API Syntax

The C syntax of the API call.

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.

REXX API Syntax

The REXX syntax of the API call, where appropriate.

A new interface, SQLDB2, has been added to support 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.

For more information about the SQLDB2 interface, see the *Embedded SQL Programming Guide*.

REXX Parameters

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

Sample Programs

The location and the names of sample programs illustrating the use of the API in one or more supported languages (C, COBOL, FORTRAN, and REXX).

Usage Notes

Other information.

See Also

A cross-reference to related information.

sqlabndx - Bind

Invokes the bind utility, which prepares SQL statements stored in the bind file generated by the precompiler, and creates a package that is stored in the database.

Scope

This API can be called from any node in `db2nodes.cfg`. It updates the database catalogs on the catalog node. Its effects are visible to all nodes.

Authorization

One of the following:

- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sql.h */
/* API: Bind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlabndx (
    _SQLOLDCHAR * pBindFileName,
    _SQLOLDCHAR * pMsgFileName,
    struct sqlopt * pBindOptions,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sql.h */
/* API: Bind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbndx (
    unsigned short MsgFileNameLen,
    unsigned short BindFileNameLen,
    struct sqlca * pSqlca,
    struct sqlopt * pBindOptions,
    _SQLOLDCHAR * pMsgFileName,
    _SQLOLDCHAR * pBindFileName);
/* ... */
```

API Parameters

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

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

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pBindOptions
Input. A structure used to pass bind options to the API. For more information about this structure, see “SQLOPT” on page 414.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pBindFileName
Input. A string containing the name of the bind file, or the name of a file containing a list of bind file names. The bind file names must contain the extension `.bnd`. A path for these files can be specified.

Precede the name of a bind list file with the at sign (@). For example, a fully qualified bind list file name might be:

```
/u/user1/bnd/@all.lst
```

The bind list file should contain one or more bind file names, and must have the extension `.lst`.

Precede all but the first bind file name with a plus symbol (+). The bind file names may be on one or more lines. For example, the bind list file `all.lst` might contain:

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

Path specifications on bind file names in the list file can be used. If no path is specified, the database manager takes path information from the bind list file.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\makeapi.sqc
COBOL	\sqllib\samples\cobol\prepbind.sqb
FORTRAN	\sqllib\samples\fortran\prepbind.sqf

Usage Notes

Binding can be done as part of the precompile process for an application program source file, or as a separate step at a later time. Use BIND when binding is performed as a separate process.

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called `myapp.sqc` generates a default bind file called `myapp.bnd` and a default package name of `MYAPP`. (However, the bind file name and the package name can be overridden at precompile time by using the **SQL_BIND_OPT** and the **SQL_PKG_OPT** options in “sqlaprep - Precompile Program” on page 18.)

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 has prerequisite requirements and restrictions beyond the scope of this manual. For more detailed information about binding application programs to databases, see the *Embedded SQL Programming Guide*.

The following table lists valid values for the *type* and the *val* fields of the bind options structure (see “SQLOPT” on page 414), as well as their corresponding CLP options.

For a description of the bind options (including default values), see the *Command Reference*.

Table 3 (Page 1 of 2). BIND Option Types and Values

CLP Option	**Option Type**	**Option Values**
ACTION ADD	SQL_ACTION_OPT	SQL_ACTION_ADD
ACTION REPLACE	SQL_ACTION_OPT	SQL_ACTION_REPLACE
BLOCKING ALL	SQL_BLOCK_OPT	SQL_BL_ALL
BLOCKING NO	SQL_BLOCK_OPT	SQL_BL_NO
BLOCKING UNAMBIG	SQL_BLOCK_OPT	SQL_BL_UNAMBIG
CCSIDG	SQL_CCSIDG_OPT	sqlopt.sqloptions.val
CCSIDM	SQL_CCSIDM_OPT	sqlopt.sqloptions.val
CCSIDS	SQL_CCSIDS_OPT	sqlopt.sqloptions.val
CHARSUB BIT	SQL_CHARSUB_OPT	SQL_CHARSUB_BIT
CHARSUB DEFAULT	SQL_CHARSUB_OPT	SQL_CHARSUB_DEFAULT
CHARSUB MIXED	SQL_CHARSUB_OPT	SQL_CHARSUB_MIXED
CHARSUB SBCS	SQL_CHARSUB_OPT	SQL_CHARSUB_SBCS
CNULREQD NO	SQL_CNULREQD_OPT	SQL_CNULREQD_NO
CNULREQD YES	SQL_CNULREQD_OPT	SQL_CNULREQD_YES
COLLECTION	SQL_COLLECTION_OPT	sqlchar structure
DATETIME DEF	SQL_DATETIME_OPT	SQL_DATETIME_DEF
DATETIME EUR	SQL_DATETIME_OPT	SQL_DATETIME_EUR
DATETIME ISO	SQL_DATETIME_OPT	SQL_DATETIME_ISO
DATETIME JIS	SQL_DATETIME_OPT	SQL_DATETIME_JIS
DATETIME LOC	SQL_DATETIME_OPT	SQL_DATETIME_LOC
DATETIME USA	SQL_DATETIME_OPT	SQL_DATETIME_USA
DECDEL COMMA	SQL_DECDEL_OPT	SQL_DECDEL_COMMA
DECDEL PERIOD	SQL_DECDEL_OPT	SQL_DECDEL_PERIOD
DEC 15	SQL_DEC_OPT	SQL_DEC_15
DEC 31	SQL_DEC_OPT	SQL_DEC_31
DEGREE 1	SQL_DEGREE_OPT	SQL_DEGREE_1
DEGREE ANY	SQL_DEGREE_OPT	SQL_DEGREE_ANY
DEGREE degree	SQL_DEGREE_OPT	Integer between 1 and 32767.
DYNAMICRULES BIND	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_BIND
DYNAMICRULES RUN	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_RUN
EXPLAIN NO	SQL_EXPLAIN_OPT	SQL_EXPLAIN_NO
EXPLAIN YES	SQL_EXPLAIN_OPT	SQL_EXPLAIN_YES
EXPLAIN ALL	SQL_EXPLAIN_OPT	SQL_EXPLAIN_ALL
EXPLSNAP NO	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_NO
EXPLSNAP YES	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_YES
EXPLSNAP ALL	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_ALL
FUNCPATH	SQL_FUNCTION_PATH	sqlchar structure

Table 3 (Page 2 of 2). BIND Option Types and Values

CLP Option	Option Type	Option Values
GENERIC	SQL_GENERIC_OPT	sqlchar structure
GRANT	SQL_GRANT_OPT	sqlchar structure
GRANT PUBLIC	SQL_GRANT_OPT	sqlchar structure
GRANT TO USER	SQL_GRANT_USER_OPT	sqlchar structure
GRANT TO GROUP	SQL_GRANT_GROUP_:OPT	sqlchar structure
INSERT BUF	SQL_INSERT_OPT	SQL_INSERT_BUF
INSERT DEF	SQL_INSERT_OPT	SQL_INSERT_DEF
ISOLATION RS	SQL_ISOLATION_OPT	SQL_ISOLATION_RS
ISOLATION NC	SQL_ISOLATION_OPT	SQL_ISOLATION_NC
ISOLATION CS	SQL_ISOLATION_OPT	SQL_ISOLATION_CS
ISOLATION RR	SQL_ISOLATION_OPT	SQL_ISOLATION_RR
ISOLATION UR	SQL_ISOLATION_OPT	SQL_ISOLATION_UR
OWNER	SQL_OWNER_OPT	sqlchar structure
QUALIFIER	SQL_QUALIFIER_OPT	sqlchar structure
QUERYOPT	SQL_QUERYOPT_OPT	SQL_QUERYOPT_0,1,2,3,5,7,9
RELEASE COMMIT	SQL_RELEASE_OPT	SQL_RELEASE_COMMIT
RELEASE DEALLOCATE	SQL_RELEASE_OPT	SQL_RELEASE_DEALLOCATE
REPLVER	SQL_REPLVER_OPT	sqlchar structure
RETAIN NO	SQL_RETAIN_OPT	SQL_RETAIN_NO
RETAIN YES	SQL_RETAIN_OPT	SQL_RETAIN_YES
SQLERROR CHECK	SQL_SQLERROR_OPT	SQL_SQLERROR_CHECK
SQLERROR CONTINUE	SQL_SQLERROR_OPT	SQL_SQLERROR_CONTINUE
SQLERROR NOPACKAGE	SQL_SQLERROR_OPT	SQL_SQLERROR_NOPACKAGE
SQLWARN NO	SQL_SQLWARN_OPT	SQL_SQLWARN_NO
SQLWARN YES	SQL_SQLWARN_OPT	SQL_SQLWARN_YES
STRDEL APOSTROPHE	SQL_STRDEL_OPT	SQL_STRDEL_APOSTROPHE
STRDEL QUOTE	SQL_STRDEL_OPT	SQL_STRDEL_QUOTE
TEXT	SQL_TEXT_OPT	sqlchar structure
VALIDATE BIND	SQL_VALIDATE_OPT	SQL_VALIDATE_BIND
VALIDATE RUN	SQL_VALIDATE_OPT	SQL_VALIDATE_RUN

Note: Option values showing `sqlchar structure` have a *val* field that contains a pointer to “SQLCHAR” on page 357. This structure contains a character string that specifies the option value.

See Also

“sqlaprep - Precompile Program” on page 18.

sqlaintp - Get Error Message

Retrieves the message associated with an error condition specified by the *sqlcode* field of the *sqlca* structure.

Authorization

None

Required Connection

None

API Include File

sql.h

C API Syntax

```
/* File: sql.h */
/* API: Get Error Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaintp (
    char * pBuffer,
    short BufferSize,
    short LineWidth,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sql.h */
/* API: Get Error Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintp (
    short BufferSize,
    short LineWidth,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pBuffer);
/* ... */
```

API Parameters

BufferSize
: Input. Size, in bytes, of a string buffer to hold the retrieved message text.

LineWidth
: Input. The maximum line width for each line of message text. Lines are broken on word boundaries. A value of zero indicates that the message text is returned without line breaks.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

pBuffer
: Output. A pointer to a string buffer where the message text is placed. If the message must be truncated to fit in the buffer, the truncation allows for the null string terminator character.

REXX API Syntax

```
GET MESSAGE INTO :msg [LINEWIDTH width]
```

REXX API Parameters

msg
: REXX variable into which the text message is placed.

width
: Maximum line width for each line in the text message. The line is broken on word boundaries. If *width* is not given or set to `0`, the message text returns without line breaks.

Sample Programs

C	\sqllib\samples\c\util.c
COBOL	\sqllib\samples\cobol\checkerr.cbl
FORTRAN	\sqllib\samples\fortran\util.f
REXX	\sqllib\samples\rexx\dbcat.cmd

Usage Notes

One message is returned per call.

A new line (line feed, LF, or carriage return/line feed, CR/LF) sequence is placed at the end of each message.

If a positive line width is specified, new line sequences are inserted between words so that the lines do not exceed the line width.

If a word is longer than a line width, the line is filled with as many characters as will fit, a new line is inserted, and the remaining characters are placed on the next line.

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.

See Also

“sqlogstt - Get SQLSTATE Message” on page 224.

sqlaprep - Precompile Program

Processes an application program source file containing embedded SQL statements. A modified source file is produced containing host language calls for the SQL statements and, by default, a package is created in the database.

Scope

This API can be called from any node in `db2nodes.cfg`. It updates the database catalogs on the catalog node. Its effects are visible to all nodes.

Authorization

One of the following:

- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sql.h */
/* API: Precompile Program */
/* ... */
SQL_API_RC SQL_API_FN
  sqlaprep (
    _SQLOLDCHAR * pProgramName,
    _SQLOLDCHAR * pMsgFileName,
    struct sqlopt * pPrepOptions,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sql.h */
/* API: Precompile Program */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgprep (
    unsigned short MsgFileNameLen,
    unsigned short ProgramNameLen,
    struct sqlca * pSqlca,
    struct sqlopt * pPrepOptions,
    _SQLOLDCHAR * pMsgFileName,
    _SQLOLDCHAR * pProgramName);
/* ... */
```

API Parameters

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

ProgramNameLen
: Input. A 2-byte unsigned integer representing the length of the program name in bytes.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pPrepOptions
: Input. A structure used to pass precompile options to the API. For more information about this structure, see “SQLOPT” on page 414.

pMsgFileName
: Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pProgramName
: Input. A string containing the name of the application to be precompiled. Use the following extensions:

  `.sqb` - for COBOL applications
  `.sqc` - for C applications
  `.sqC` - for UNIX C++ applications
  `.sqf` - for FORTRAN applications
  `.sqx` - for C++ applications

  When the TARGET option is used, the input file name extension does not have to be from this predefined list.

sqlaprep - Precompile Program

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\makeapi.sqc
COBOL	\sqllib\samples\cobol\prepbind.sqb
FORTRAN	\sqllib\samples\fortran\prepbind.sqf

Usage Notes

A modified source file is produced, which contains host language equivalents to the SQL statements. By default, a package is created in the database to which a connection has been established. The name of the package is the same as the program file name (minus the extension and folded to uppercase), up to a maximum of 8 characters.

Following connection to a database, **sqlaprep** executes under the transaction that was started. PRECOMPILE PROGRAM then issues a COMMIT or a ROLLBACK operation to terminate the current transaction and start another one.

Precompiling stops if a fatal error or more than 100 errors occur. If a fatal error does occur, PRECOMPILE PROGRAM stops precompiling, attempts to close all files, and discards the package.

The following table lists valid values for the *type* and the *val* fields of the precompile options structure (see "SQLOPT" on page 414), as well as their corresponding CLP options. For a description of the precompile options (including default values), see the *Command Reference*.

Table 4 (Page 1 of 3). PRECOMPILE Option Types and Values

CLP Option	API Option Type	API Option Values
ACTION ADD	SQL_ACTION_OPT	SQL_ACTION_ADD
ACTION REPLACE	SQL_ACTION_OPT	SQL_ACTION_REPLACE
BINDFILE	SQL_BIND_OPT	Null
BINDFILE filename	SQL_BIND_OPT	sqlchar structure
BLOCKING ALL	SQL_BLOCK_OPT	SQL_BL_ALL
BLOCKING NO	SQL_BLOCK_OPT	SQL_BL_NO
BLOCKING UNAMBIG	SQL_BLOCK_OPT	SQL_BL_UNAMBIG
CCSIDG value	SQL_CCSIDG_OPT	sqlopt.sqloptions.val
CCSIDM value	SQL_CCSIDM_OPT	sqlopt.sqloptions.val
CCSIDS value	SQL_CCSIDS_OPT	sqlopt.sqloptions.val
CHARSUB BIT	SQL_CHARSUB_OPT	SQL_CHARSUB_BIT
CHARSUB DEFAULT	SQL_CHARSUB_OPT	SQL_CHARSUB_DEFAULT
CHARSUB MIXED	SQL_CHARSUB_OPT	SQL_CHARSUB_MIXED
CHARSUB SBCS	SQL_CHARSUB_OPT	SQL_CHARSUB_SBCS
COLLECTION coll-id	SQL_COLLECTION_OPT	sqlchar structure

Table 4 (Page 2 of 3). PRECOMPILE Option Types and Values

CLP Option	API Option Type	API Option Values
CONNECT 1	SQL_CONNECT_OPT	SQL_CONNECT_1
CONNECT 2	SQL_CONNECT_OPT	SQL_CONNECT_2
DATETIME DEF	SQL_DATETIME_OPT	SQL_DATETIME_DEF
DATETIME EUR	SQL_DATETIME_OPT	SQL_DATETIME_EUR
DATETIME ISO	SQL_DATETIME_OPT	SQL_DATETIME_ISO
DATETIME JIS	SQL_DATETIME_OPT	SQL_DATETIME_JIS
DATETIME LOC	SQL_DATETIME_OPT	SQL_DATETIME_LOC
DATETIME USA	SQL_DATETIME_OPT	SQL_DATETIME_USA
DECDEL COMMA	SQL_DECDEL_OPT	SQL_DECDEL_COMMA
DECDEL PERIOD	SQL_DECDEL_OPT	SQL_DECDEL_PERIOD
DEC 15	SQL_DEC_OPT	SQL_DEC_15
DEC 31	SQL_DEC_OPT	SQL_DEC_31
DEFERRED_PREPARE ALL	SQL_DEFERRED_PREPARE_OPT	SQL_DEFERRED_PREPARE_ALL
DEFERRED_PREPARE NO	SQL_DEFERRED_PREPARE_OPT	SQL_DEFERRED_PREPARE_NO
DEFERRED_PREPARE YES	SQL_DEFERRED_PREPARE_OPT	SQL_DEFERRED_PREPARE_YES
DEGREE 1	SQL_DEGREE_OPT	SQL_DEGREE_1
DEGREE ANY	SQL_DEGREE_OPT	SQL_DEGREE_ANY
DEGREE degree	SQL_DEGREE_OPT	Integer between 1 and 32767.
DISCONNECT EXPLICIT	SQL_DISCONNECT_OPT	SQL_DISCONNECT_EXPL
DISCONNECT CONDITIONAL	SQL_DISCONNECT_OPT	SQL_DISCONNECT_COND
DISCONNECT AUTOMATIC	SQL_DISCONNECT_OPT	SQL_DISCONNECT_AUTO
DYNAMICRULES BIND	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_BIND
DYNAMICRULES RUN	SQL_DYNAMICRULES_OPT	SQL_DYNAMICRULES_RUN
EXPLAIN NO	SQL_EXPLAIN_OPT	SQL_EXPLAIN_NO
EXPLAIN YES	SQL_EXPLAIN_OPT	SQL_EXPLAIN_YES
EXPLAIN ALL	SQL_EXPLAIN_OPT	SQL_EXPLAIN_ALL Not supported by DRDA.
EXPLSNAP NO	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_NO
EXPLSNAP YES	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_YES
EXPLSNAP ALL	SQL_EXPLSNAP_OPT	SQL_EXPLSNAP_ALL
FUNCPATH	SQL_FUNCTION_PATH	sqlchar structure
INSERT BUF	SQL_INSERT_OPT	SQL_INSERT_BUF
INSERT DEF	SQL_INSERT_OPT	SQL_INSERT_DEF
ISOLATION RS	SQL_ISOLATION_OPT	SQL_ISOLATION_RS
ISOLATION NC	SQL_ISOLATION_OPT	SQL_ISOLATION_NC
ISOLATION CS	SQL_ISOLATION_OPT	SQL_ISOLATION_CS
ISOLATION RR	SQL_ISOLATION_OPT	SQL_ISOLATION_RR
ISOLATION UR	SQL_ISOLATION_OPT	SQL_ISOLATION_UR
LANGLEVEL SAA1	SQL_STANDARDS_OPT	SQL_SAA_COMP
LANGLEVEL MIA	SQL_STANDARDS_OPT	SQL_MIA_COMP
LANGLEVEL SQL92E	SQL_STANDARDS_OPT	SQL_SQL92E_COMP
LEVEL levelname	SQL_LEVEL_OPT	sqlchar structure
NOLINEMACRO	SQL_LINEMACRO_OPT	SQL_NO_LINE_MACROS
(default)	SQL_LINEMACRO_OPT	SQL_LINE_MACROS
OPTLEVEL 0	SQL_OPTIM_OPT	SQL_DONT_OPTIMIZE
OPTLEVEL 1	SQL_OPTIM_OPT	SQL_OPTIMIZE

Table 4 (Page 3 of 3). PRECOMPILE Option Types and Values

CLP Option	API Option Type	API Option Values
OUTPUT filename	SQL_PREP_OUTPUT_OPT	sqlchar structure
OWNER	SQL_OWNER_OPT	sqlchar structure
PACKAGE	SQL_PKG_OPT	Null
PACKAGE pkgname	SQL_PKG_OPT	sqlchar structure
QUALIFIER	SQL_QUALIFIER_OPT	sqlchar structure
QUERYOPT	SQL_QUERYOPT_OPT	SQL_QUERYOPT_0,1,2,3,5,7,9
RELEASE COMMIT	SQL_RELEASE_OPT	SQL_RELEASE_COMMIT
RELEASE DEALLOCATE	SQL_RELEASE_OPT	SQL_RELEASE_DEALLOCATE
REPLVER versn-str	SQL_REPLVER_OPT	sqlchar structure
RETAIN NO	SQL_RETAIN_OPT	SQL_RETAIN_NO
RETAIN YES	SQL_RETAIN_OPT	SQL_RETAIN_YES
SQLCA SAA	SQL_SAA_OPT	SQL_SAA_YES
SQLCA NONE	SQL_SAA_OPT	SQL_SAA_NO
SQLERROR CHECK	SQL_SQLERROR_OPT	SQL_SQLERROR_CHECK
SQLERROR CONTINUE	SQL_SQLERROR_OPT	SQL_SQLERROR_CONTINUE
SQLERROR NOPACKAGE	SQL_SQLERROR_OPT	SQL_SQLERROR_NOPACKAGE
SQLFLAG SQL92E SYNTAX	SQL_FLAG_OPT	SQL_SQL92E_SYNTAX
SQLFLAG MVSDB2V23 SYNTAX	SQL_FLAG_OPT	SQL_MVSDB2V23_SYNTAX
SQLFLAG MVSDB2V31 SYNTAX	SQL_FLAG_OPT	SQL_MVSDB2V31_SYNTAX
SQLFLAG MVSDB2V41 SYNTAX	SQL_FLAG_OPT	SQL_MVSDB2V41_SYNTAX
SQLRULES DB2	SQL_RULES_OPT	SQL_RULES_DB2
SQLRULES STD	SQL_RULES_OPT	SQL_RULES_STD
SQLWARN NO	SQL_SQLWARN_OPT	SQL_SQLWARN_NO
SQLWARN YES	SQL_SQLWARN_OPT	SQL_SQLWARN_YES
STRDEL APOSTROPHE	SQL_STRDEL_OPT	SQL_STRDEL_APOSTROPHE
STRDEL QUOTE	SQL_STRDEL_OPT	SQL_STRDEL_QUOTE
SYNCPOINT ONEPHASE	SQL_SYNCPOINT_OPT	SQL_SYNC_ONEPHASE
SYNCPOINT TWOPHASE	SQL_SYNCPOINT_OPT	SQL_SYNC_TWOPHASE
SYNCPOINT NONE	SQL_SYNCPOINT_OPT	SQL_SYNC_NONE
SYNTAX	SQL_SYNTAX_OPT	SQL_SYNTAX_CHECK
(default)	SQL_SYNTAX_OPT	SQL_NO_SYNTAX_CHECK
TARGET compiler	SQL_TARGET_OPT	sqlchar structure
TEXT text-str	SQL_TEXT_OPT	sqlchar structure
VALIDATE BIND	SQL_VALIDATE_OPT	SQL_VALIDATE_BIND
VALIDATE RUN	SQL_VALIDATE_OPT	SQL_VALIDATE_RUN
VERSION versn-str	SQL_VERSION_OPT	sqlchar structure
WCHARTYPE CONVERT	SQL_WCHAR_OPT	SQL_WCHAR_CONVERT
WCHARTYPE NOCONVERT	SQL_WCHAR_OPT	SQL_WCHAR_NOCONVERT
(none)	SQL_NO_OPT	(none)

See Also

"sqlabndx - Bind" on page 10.

sqlarbnd - Rebind

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization

One of the following:

- *sysadm* or *dbadm* authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

The authorization ID logged in the BOUNDBY column of the SYSCAT.PACKAGES system catalog table, which is the ID of the most recent binder of the package, is used as the binder authorization ID for the rebind, and for the default *schema* for table references in the package. Note that this default qualifier may be different from the authorization ID of the user executing the rebind request. REBIND will use the same bind options that were specified when the package was created.

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sql.h */
/* API: Rebind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlarbnd (
    char * pPackageName,
    struct sqlca * pSqlca,
    void * pReserved);
/* ... */
```

sqlarbnd - Rebind

Generic API Syntax

```
/* File: sql.h */
/* API: Rebind */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrbnd (
    unsigned short PackageNameLen,
    char * pPackageName,
    struct sqlca * pSqlca,
    void * pReserved);
/* ... */
```

API Parameters

PackageNameLen
Input. A 2-byte unsigned integer representing the length of the package name in bytes.

pPackageName
Input. A string containing the qualified or unqualified name that designates the package to be rebound. An unqualified package name is implicitly qualified by the current authorization ID.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pReserved
Reserved for future use. Must be set to NULL.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\rebind.sqc
COBOL	\sqllib\samples\cobol\rebind.sqb
FORTRAN	\sqllib\samples\fortran\rebind.sqf

Usage Notes

REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables "what if" analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. It also permits multiple rebinds within a unit of work.

This API:

- Provides a quick way to recreate a package. This enables the user to take advantage of a change in the system without a need for the original bind file. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, REBIND can be used to recreate the package. REBIND can also be used to recreate packages after "sqlustat - Runstats" on page 319 has been executed, thereby taking advantage of the new statistics.
- Provides a method to recreate inoperative packages. Inoperative packages must be explicitly rebound by invoking either the bind utility or the rebind utility. A package will be marked inoperative (the VALID column of the `SYSCAT.PACKAGES` system catalog will be set to `X`) if a function instance on which the package depends is dropped.
- Gives users control over the rebinding of invalid packages. Invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This may result in a noticeable delay in the execution of the first SQL request for the invalid package. It may be desirable to explicitly rebind invalid packages, rather than allow the system to automatically rebind them, in order to eliminate the initial delay and to prevent unexpected SQL error messages which may be returned in case the implicit rebind fails. For example, following migration, all packages stored in the database will be invalidated by the DB2 Version 2.1 migration process. Given that this may involve a large number of packages, it may be desirable to explicitly rebind all of the invalid packages at one time. This explicit rebinding can be accomplished using BIND, REBIND, or the **db2rbind** tool (see "db2rbind - Rebind all Packages" in the *Command Reference*).

The choice of whether to use BIND or REBIND to explicitly rebind a package depends on the circumstances. It is recommended that REBIND be used whenever the situation does not specifically require the use of BIND, since the performance of REBIND is significantly better than that of BIND. BIND *must* be used, however:

- When there have been modifications to the program (for example, when SQL statements have been added or deleted, or when the package does not match the executable for the program).
- When the user wishes to modify any of the bind options as part of the rebind. REBIND does not support any bind options. For example, if the user wishes to have privileges on the package granted as part of the bind process, BIND must be used, since it has an **SQL_GRANT_OPT** option.
- When the package does not currently exist in the database.
- When detection of *all* bind errors is desired. REBIND only returns the first error it detects, and then ends, whereas the BIND command returns the first 100 errors that occur during binding.

REBIND is supported by DDCS.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the SYSCAT.PACKAGES system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the SYSCAT.STATEMENTS system catalog table.

If 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.

See Also

"sqlabndx - Bind" on page 10
"sqlustat - Runstats" on page 319.

sqlbctcq - Close Tablespace Container Query

Ends a table space container query request and frees the associated resources.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Close Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctcq (
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Close Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctcq (
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqlbctcq - Close Tablespace Container Query

Sample Programs

C	\sqllib\samples\c\tabscont.sqc
COBOL	\sqllib\samples\cobol\tabscont.sqb
FORTRAN	\sqllib\samples\fortran\tabscont.sqf

See Also

"sqlbftcq - Fetch Tablespace Container Query" on page 31
"sqlbotcq - Open Tablespace Container Query" on page 42
"sqlbstsc - Set Tablespace Containers" on page 51
"sqlbtcq - Tablespace Container Query" on page 54.

sqlbctsq - Close Tablespace Query

Ends a table space query request, and frees up associated resources.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Close Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbctsq (
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Close Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctsq (
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqlbctsq - Close Tablespace Query

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tabspace.sqb
FORTRAN	\sqllib\samples\fortran\tabspace.sqf

See Also

"sqlbftpq - Fetch Tablespace Query" on page 34
"sqlbgtss - Get Tablespace Statistics" on page 37
"sqlbotsq - Open Tablespace Query" on page 45
"sqlbstpq - Single Tablespace Query" on page 48
"sqlbmtsq - Tablespace Query" on page 39.

sqlbftcq - Fetch Tablespace Container Query

Fetches a specified number of rows of table space container query data, each row consisting of data for a container.

Scope

In a partitioned database server environment, only the table spaces on the current node are listed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftcq (
    struct sqlca * pSqlca,
    unsigned long MaxContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData,
    unsigned long * pNumContainers);
/* ... */
```

sqlbftcq - Fetch Tablespace Container Query

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftcq (
    struct sqlca * pSqlca,
    unsigned long MaxContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData,
    unsigned long * pNumContainers);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

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" on page 349. The caller of this API must allocate space for *MaxContainers* of these structures, and set *pContainerData* to point to this space. The API will use this space to return the table space container data.

pNumContainers
: Output. Number of rows of output returned.

Sample Programs

C	\sqllib\samples\c\tabscont.sqc
COBOL	\sqllib\samples\cobol\tabscont.sqb
FORTRAN	\sqllib\samples\fortran\tabscont.sqf

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**.

For more information, see "sqlbotcq - Open Tablespace Container Query" on page 42.

See Also

“sqlbctcq - Close Tablespace Container Query” on page 27
“sqlbotcq - Open Tablespace Container Query” on page 42
“sqlbstsc - Set Tablespace Containers” on page 51
“sqlbtcq - Tablespace Container Query” on page 54.

sqlbftpq - Fetch Tablespace Query

Fetches a specified number of rows of table space query data, each row consisting of data for a table space.

Scope

In a partitioned database server environment, only the table spaces on the current node are listed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbftpq (
    struct sqlca * pSqlca,
    unsigned long MaxTablespaces,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    unsigned long * pNumTablespaces);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Fetch Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgftpq (
    struct sqlca * pSqlca,
    unsigned long MaxTablespaces,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    unsigned long * pNumTablespaces);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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” on page 351. 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.

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tabspace.sqb
FORTRAN	\sqllib\samples\fortran\tabspace.sqf

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**.

For more information, see “sqlbotsq - Open Tablespace Query” on page 45.

See Also

“sqlbctsq - Close Tablespace Query” on page 29
“sqlbgtss - Get Tablespace Statistics” on page 37
“sqlbotsq - Open Tablespace Query” on page 45
“sqlbstpq - Single Tablespace Query” on page 48
“sqlbmtsq - Tablespace Query” on page 39.

sqlbgtss - Get Tablespace Statistics

Provides information on the space utilization of a table space.

Scope

In a partitioned database server environment, only the table spaces on the current node are listed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Tablespace Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbgtss (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    struct SQLB_TBS_STATS * pTablespaceStats);
/* ... */
```

sqlbgtss - Get Tablespace Statistics

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Tablespace Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtss (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    struct SQLB_TBS_STATS * pTablespaceStats);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

TablespaceId
: Input. ID of the single table space to be queried.

pTablespaceStats
: Output. A pointer to a user-allocated *SQLB_TBS_STATS* structure. The information about the table space is returned in this structure. For more information about this structure, see “SQLB-TBS-STATS” on page 347.

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tabspace.sqb
FORTRAN	\sqllib\samples\fortran\tabspace.sqf

Usage Notes

See “SQLB-TBS-STATS” on page 347 for information about the fields returned and their meaning.

See Also

“sqlbctsq - Close Tablespace Query” on page 29
“sqlbftpq - Fetch Tablespace Query” on page 34
“sqlbotsq - Open Tablespace Query” on page 45
“sqlbstpq - Single Tablespace Query” on page 48
“sqlbmtsq - Tablespace Query” on page 39.

sqlbmtsq - Tablespace Query

Provides a one-call interface to the table space query data. The query data for all table spaces in the database is returned in an array.

Scope

In a partitioned database server environment, only the table spaces on the current node are listed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbmtsq (
    struct sqlca * pSqlca,
    unsigned long * pNumTablespaces,
    struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
    unsigned long reserved1,
    unsigned long reserved2);
/* ... */
```

sqlbmtsq - Tablespace Query

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmtsq (
    struct sqlca * pSqlca,
    unsigned long * pNumTablespaces,
    struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
    unsigned long reserved1,
    unsigned long reserved2);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

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`.

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tabspace.sqb
FORTRAN	\sqllib\samples\fortran\tabspace.sqf

Usage Notes

This API uses the lower level services, namely:

- "sqlbotsq - Open Tablespace Query" on page 45
- "sqlbftpq - Fetch Tablespace Query" on page 34
- "sqlbctsq - Close Tablespace Query" on page 29

to get all of the table space query data at once.

If sufficient memory is available, this function returns the number of table spaces, and a pointer to the memory location of the table space query data. It is the user's responsibility to free this memory with a call to **sqlefmem** (see “sqlefmem - Free Memory” on page 117).

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 - Open Tablespace Query” on page 45, “sqlbftpq - Fetch Tablespace Query” on page 34, and “sqlbctsq - Close Tablespace Query” on page 29, to fetch less than the whole list at once.

See Also

“sqlbctsq - Close Tablespace Query” on page 29
“sqlbftpq - Fetch Tablespace Query” on page 34
“sqlbgtss - Get Tablespace Statistics” on page 37
“sqlbotsq - Open Tablespace Query” on page 45
“sqlbstpq - Single Tablespace Query” on page 48.

sqlbotcq - Open Tablespace Container Query

Prepares for a table space container query operation, and returns the number of containers currently in the table space.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Open Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotcq (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    unsigned long * pNumContainers);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Open Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotcq (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    unsigned long * pNumContainers);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

TablespaceId
: Input. ID of the table space for which container data is desired. If the special identifier `SQLB_ALL_TABLESPACES` (in `sqlutil`) is specified, a complete list of containers for the entire database is produced.

pNumContainers
: Output. The number of containers in the specified table space.

Sample Programs

C	\sqllib\samples\c\tabscont.sqc
COBOL	\sqllib\samples\cobol\tabscont.sqb
FORTRAN	\sqllib\samples\fortran\tabscont.sqf

Usage Notes

This API is normally followed by one or more calls to “sqlbftcq - Fetch Tablespace Container Query” on page 31, and then by one call to “sqlbctcq - Close Tablespace Container Query” on page 27.

An application can use the following APIs to fetch information about containers in use by table spaces:

- “sqlbtcq - Tablespace Container Query” on page 54

  Fetches a complete list of container information. The API allocates the space required to hold the information for all the containers, and returns a pointer to this information. Use this API to scan the list of containers for specific information. Using this API is identical to calling the three APIs below (**sqlbotcq**, **sqlbftcq**, 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 - Free Memory” on page 117 to free the memory.
- “sqlbotcq - Open Tablespace Container Query” on page 42
- “sqlbftcq - Fetch Tablespace Container Query” on page 31
- “sqlbctcq - Close Tablespace Container Query” on page 27

  These three APIs function like an SQL cursor, in that they use the OPEN/FETCH/CLOSE paradigm. The caller must provide the output area for the fetch. Unlike an SQL cursor, only one table space container query can be active at a time. Use this set of APIs to scan the list of table space containers for specific information. These APIs allows the user to control the memory requirements of an application (compared with “sqlbtcq - Tablespace Container Query” on page 54).

When **sqlbotcq** is called, a snapshot of the current container information is formed in the agent servicing the application. If the application issues a second table space container query call (**sqlbtcq** or **sqlbotcq**), this snapshot is replaced with refreshed information.

No locking is performed, so the information in the buffer may not reflect changes made by another application after the snapshot was generated. The information is not part of a transaction.

There is one snapshot buffer for table space queries and another for table space container queries. These buffers are independent of one another.

See Also

"sqlbctcq - Close Tablespace Container Query" on page 27
"sqlbftcq - Fetch Tablespace Container Query" on page 31
"sqlbstsc - Set Tablespace Containers" on page 51
"sqlbtcq - Tablespace Container Query" on page 54.

sqlbotsq - Open Tablespace Query

Prepares for a table space query operation, and returns the number of table spaces currently in the database.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Open Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbotsq (
    struct sqlca * pSqlca,
    unsigned long TablespaceQueryOptions,
    unsigned long * pNumTablespaces);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Open Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgotsq (
    struct sqlca * pSqlca,
    unsigned long TablespaceQueryOptions,
    unsigned long * pNumTablespaces);
/* ... */
```

sqlbotsq - Open Tablespace Query

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

TablespaceQueryOptions
: Input. Indicates which table spaces to process. Valid values (defined in `sqlutil`) are:

  SQLB_OPEN_TBS_ALL
  : Process all the table spaces in the database.

  SQLB_OPEN_TBS_RESTORE
  : Process only the table spaces that the user's agent is restoring.

pNumTablespaces
: Output. The number of table spaces in the connected database.

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tabspace.sqb
FORTRAN	\sqllib\samples\fortran\tabspace.sqf

Usage Notes

This API is normally followed by one or more calls to "sqlbftpq - Fetch Tablespace Query" on page 34, and then by one call to "sqlbctsq - Close Tablespace Query" on page 29.

An application can use the following APIs to fetch information about the currently defined table spaces:

- "sqlbstpq - Single Tablespace Query" on page 48

  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 - Tablespace Query" on page 39

  Fetches information about all table spaces. The API allocates the space required to hold the information for all table spaces, and returns a pointer to this information. Use this API to scan the list of table spaces when searching for specific information. Using this API is identical to calling the three APIs below, except that this API automatically allocates the memory for the output information. A call to this API must be followed by a call to "sqlefmem - Free Memory" on page 117 to free the memory.

- "sqlbotsq - Open Tablespace Query" on page 45
- "sqlbftpq - Fetch Tablespace Query" on page 34
- "sqlbctsq - Close Tablespace Query" on page 29

  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 - Tablespace Query" on page 39).

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 (**sqlbtsq** 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.

See Also

"sqlbctsq - Close Tablespace Query" on page 29
"sqlbftpq - Fetch Tablespace Query" on page 34
"sqlbstpq - Single Tablespace Query" on page 48
"sqlbmtsq - Tablespace Query" on page 39.

sqlbstpq - Single Tablespace Query

Retrieves information about a single currently defined table space.

Scope

In a partitioned database server environment, only the table spaces on the current node are listed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Single Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstpq (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    unsigned long reserved);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Single Tablespace Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstpq (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    struct SQLB_TBSPQRY_DATA * pTablespaceData,
    unsigned long reserved);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

TablespaceId
: Input. Identifier for the table space which is to be queried.

pTablespaceData
: Input and output. Pointer to a user-supplied *SQLB_TBSPQRY_DATA* structure where the table space information will be placed upon return. The caller of this API must initialize the structure and set TBSPQVER to `SQLB_TBSPQRY_DATA_ID` (in `sqlutil`).

reserved
: Input. Always `SQLB_RESERVED1`.

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tabspace.sqb
FORTRAN	\sqllib\samples\fortran\tabspace.sqf

Usage Notes

This API retrieves information about a single table space if the table space identifier to be queried is known. This API provides an alternative to the more expensive OPEN TABLESPACE QUERY, FETCH, and CLOSE combination of APIs, which must be used to scan for the desired table space when the table space identifier is not known in advance. The table space IDs can be found in the system catalogs. No agent snapshot is taken; since there is only one entry to return, it is returned directly.

For more information, see “sqlbotsq - Open Tablespace Query” on page 45.

sqlbstpq - Single Tablespace Query

See Also

“sqlbctsq - Close Tablespace Query” on page 29
“sqlbftpq - Fetch Tablespace Query” on page 34
“sqlbgtss - Get Tablespace Statistics” on page 37
“sqlbotsq - Open Tablespace Query” on page 45
“sqlbmtsq - Tablespace Query” on page 39.

sqlbstsc - Set Tablespace Containers

This API facilitates the provision of a *redirected* restore, in which the user is restoring a database, and a different set of operating system storage containers is desired or required.

Use this API when the table space is in a *storage definition pending* or a *storage definition allowed* state. These states are possible during a restore operation, immediately prior to the restoration of database pages.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Set Tablespace Containers */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbstsc (
    struct sqlca * pSqlca,
    unsigned long SetContainerOptions,
    unsigned long TablespaceId,
    unsigned long NumContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData);
/* ... */
```

sqlbstsc - Set Tablespace Containers

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Set Tablespace Containers */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstsc (
    struct sqlca * pSqlca,
    unsigned long SetContainerOptions,
    unsigned long TablespaceId,
    unsigned long NumContainers,
    struct SQLB_TBSCONTQRY_DATA * pContainerData);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

SetContainerOptions
: Input. Use this field to specify additional options. Valid values (defined in `sqlutil`) are:

  SQLB_SET_CONT_INIT_STATE
  Redo alter table space operations when performing a roll forward.

  SQLB_SET_CONT_FINAL_STATE
  Ignore alter table space operations in the log when performing a roll forward.

TablespaceId
: Input. Identifier for the table space which is to be changed.

NumContainers
: Input. The number of rows the structure pointed to by *pContainerData* holds.

pContainerData
: Input. Container specifications. Although the *SQLB_TBSCONTQRY_DATA* structure is used, only the *contType*, *totalPages*, *name*, and *nameLen* (for languages other than C) fields are used; all other fields are ignored.

Sample Programs

C	\sqllib\samples\c\backrest.c
COBOL	\sqllib\samples\cobol\backrest.cbl
FORTRAN	\sqllib\samples\fortran\backrest.f

Usage Notes

This API is used in conjunction with “sqlurst - Restore Database” on page 309.

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. For more information, see the *Administration Guide*.

Typical use of this API would involve the following sequence of actions:

1. Invoke “sqlurst - Restore Database” on page 309 with *CallerAction* set to SQLUD_RESTORE_STORDEF.

   The restore utility returns an *sqlcode* indicating that some of the containers are inaccessible.

2. Invoke **sqlbstsc** to set the table space container definitions with the *SetContainerOptions* parameter set to SQLB_SET_CONT_FINAL_STATE.

3. Invoke **sqlurst** a second time with *CallerAction* set to SQLUD_CONTINUE.

The above sequence will allow the restore to use the new table space container definitions and will ignore table space add container operations in the logs when “sqluroll - Rollforward Database” on page 300 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/rollforward to replace all of the original data into these new containers. If there is not sufficient space, then such tablespaces 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/rollforward is needed, he will know how much disk space is required.

See Also

“sqlubkp - Backup Database” on page 230
“sqluroll - Rollforward Database” on page 300
“sqlurst - Restore Database” on page 309.

sqlbtcq - Tablespace Container Query

Provides a one-call interface to the table space container query data. The query data for all containers in a table space, or for all containers in all table spaces, is returned in an array.

Scope

In a partitioned database server environment, only the table spaces on the current node are listed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlbtcq (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    unsigned long * pNumContainers,
    struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Tablespace Container Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgtcq (
    struct sqlca * pSqlca,
    unsigned long TablespaceId,
    unsigned long * pNumContainers,
    struct SQLB_TBSCONTQRY_DATA ** ppContainerData);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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.

Sample Programs

C	\sqllib\samples\c\tabscont.sqc
COBOL	\sqllib\samples\cobol\tabscont.sqb
FORTRAN	\sqllib\samples\fortran\tabscont.sqf

Usage Notes

This API uses the lower level services, namely:

- “sqlbotcq - Open Tablespace Container Query” on page 42
- “sqlbftcq - Fetch Tablespace Container Query” on page 31
- “sqlbctcq - Close Tablespace Container Query” on page 27

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** (see “sqlefmem - Free Memory” on page 117).

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 - Open Tablespace Container Query” on page 42, “sqlbftcq - Fetch Tablespace Container Query” on page 31, and “sqlbctcq - Close Tablespace Container Query” on page 27 to fetch less than the whole list at once.

See Also

“sqlbctcq - Close Tablespace Container Query” on page 27
“sqlbftcq - Fetch Tablespace Container Query” on page 31
“sqlbotcq - Open Tablespace Container Query” on page 42
“sqlbstsc - Set Tablespace Containers” on page 51
“sqlbtcq - Tablespace Container Query” on page 54.

sqlcspqy - List DRDA Indoubt Transactions

Provides a list of transactions that are indoubt between partner LUs connected by LU 6.2 protocols.

Authorization

sysadm

Required Connection

Instance

API Include File

sqlxa.h

C API Syntax

```
/* File: sqlxa.h */
/* API: List DRDA Indoubt Transactions */
/* ... */
extern int SQL_API_FN sqlcspqy(SQLCSPQY_INDOUBT     **indoubt_data,
                               long                  *indoubt_count,
                               struct sqlca          *sqlca);
/* ... */
```

API Parameters

indoubt_data
: Output. A pointer to the returned array.

indoubt_count
: Output. The number of elements in the returned array.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Usage Notes

DRDA indoubt transactions occur when communication is lost between coordinators and participants in distributed units of work.

A distributed unit of work lets a user or application read and update data at multiple locations within a single unit of work. Such work requires a two-phase commit.

The first phase requests all the participants to prepare for commit. The second phase commits or rolls back the transactions. If a coordinator or participant becomes unavailable after the first phase then the distributed transactions are indoubt.

Before issuing LIST DRDA INDOUBT TRANSACTIONS, the application process must be connected to the Sync Point Manager (SPM) instance. Use the SPM_NAME as the *dbalias* on the CONNECT statement (see the *SQL Reference* for more information about using CONNECT). SPM_NAME is a database manager configuration parameter.

sqle_activate_db - Activate Database

Activates the specified database and starts up all necessary database services, so that the database is available for connection and use by any application.

Scope

This API activates the specified database on all nodes within the system. If one or more of these nodes encounters an error during activation of the database, a warning is returned. The database remains activated on all nodes on which the API has succeeded.

Note: If it is the coordinator node or the catalog node that encounters the error, the API returns a negative *sqlcode*, and the database will not be activated on any node.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

None. Applications invoking ACTIVATE DATABASE cannot have any existing database connections.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Activate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_activate_db (
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

sqle_activate_db - Activate Database

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Activate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_activate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pDbAlias
: Input. Pointer to the database alias name.

pUserName
: Input. Pointer to the user ID starting the database. Can be NULL.

pPassword
: Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

If a database has not been started, and a DB2 CONNECT TO (or an implicit connect) is encountered in an application, the application must wait while the database manager starts up the required database. In such cases, this first application spends time on database initialization before it can do any work. However, once the first application has started a database, other applications can simply connect and use it.

Database administrators can use ACTIVATE DATABASE to start up selected databases. This eliminates any application time spent on database initialization.

Databases initialized by ACTIVATE DATABASE can only be shut down by "sqle_deactivate_db - Deactivate Database" on page 62, or by "sqlepstp - Stop Database Manager" on page 159. To obtain a list of activated databases, call "sqlmonss - Get Snapshot" on page 215.

If a database was started by a DB2 CONNECT TO (or an implicit connect) and subsequently an ACTIVATE DATABASE is issued for that same database, then DEACTIVATE DATABASE must be used to shut down that database.

ACTIVATE DATABASE behaves in a similar manner to a DB2 CONNECT TO (or an implicit connect) when working with a database requiring a restart (for example, database in an inconsistent state). The database will be restarted before it can be initialized by ACTIVATE DATABASE.

See Also

"sqle_deactivate_db - Deactivate Database" on page 62.

sqle_deactivate_db - Deactivate Database

Stops the specified database.

Scope

In an MPP system, this API deactivates the specified database on all nodes in the system. If one or more of these nodes encounters an error, a warning is returned. The database will be successfully deactivated on some nodes, but may remain activated on the nodes encountering the error.

Note: If it is the coordinator node or the catalog node that encounters the error, the API returns a negative *sqlcode*, and the database will not be reactivated on any node on which it was deactivated.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

None. Applications invoking DEACTIVATE DATABASE cannot have any existing database connections.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Deactivate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Deactivate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlg_deactivate_db (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    char * pDbAlias,
    char * pUserName,
    char * pPassword,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pDbAlias
: Input. Pointer to the database alias name.

pUserName
: Input. Pointer to the user ID stopping the database. Can be NULL.

pPassword
: Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

Databases initialized by ACTIVATE DATABSE can only be shut down by DEACTIVATE DATABASE. “sqlepstp - Stop Database Manager” on page 159 automatically stops all activated databases before stopping the database manager. If a database was initialized by ACTIVATE DATABASE, the last DB2 CONNECT RESET statement (counter equal 0) will not shut down the database; DEACTIVATE DATABASE must be used.

sqle_deactivate_db - Deactivate Database

See Also

“sqle_activate_db - Activate Database” on page 59.

sqleaddn - Add Node

Adds a new node to the parallel database system. This API creates database partitions for all databases currently defined in the MPP server on the new node. The user can specify the source node for any temporary table spaces to be created with the databases, or specify that no temporary table spaces are to be created. The API must be issued from the node that is being added, and can only be issued on an MPP server.

Scope

This API only affects the node on which it is executed.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Add Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqleaddn (
    void * pAddNodeOptions,
    struct sqlca * pSqlca);
/* ... */
```

sqleaddn - Add Node

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Add Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddn (
    unsigned short addnOptionsLen,
    struct sqlca * pSqlca,
    void * pAddNodeOptions);
/* ... */
```

API Parameters

addnOptionsLen
: Input. A 2-byte unsigned integer representing the length of the optional *sqle_addn_options* structure in bytes.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

pAddNodeOptions
: Input. A pointer to the optional *sqle_addn_options* structure. This structure is used to specify the source node, if any, of the temporary table space definitions for all database partitions created during the add node operation. If not specified (that is, a NULL pointer is specified), the temporary table space definitions will be the same as those for the catalog node. For more information about this structure, see "SQLE-ADDN-OPTIONS" on page 365.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

Before adding a new node, ensure that there is sufficient storage for the containers that must be created for all existing databases on the system.

The add node operation creates an empty database partition on the new node for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default value.

If an add node operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the node being added (that is, the local node). Existing database partitions remain unaffected on all other nodes. If this fails, no further clean up is done, and an error is returned.

The database partitions on the new node cannot be used to contain user data until after the ALTER NODEGROUP statement has been used to add the node to a nodegroup. For details, see the *SQL Reference*.

This API will fail if a create database or a drop database operation is in progress. The API can be called again once the operation has completed.

If temporary table spaces are to be created with the database partitions, **sqleaddn** may have to communicate with another node in the MPP system 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 node must respond with the table space definitions. If this time is exceeded, the API fails. Increase the value of *start_stop_time*, and call the API again.

See Also

"sqlecrea - Create Database" on page 81
"sqledrpn - Drop Node Verify" on page 113
"sqlepstart - Start Database Manager" on page 156.

sqleatin - Attach

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the **DB2INSTANCE** environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Authorization

None

Required Connection

This API establishes an instance attachment.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Attach */
/* ... */
SQL_API_RC SQL_API_FN
  sqleatin (
    char * pNodeName,
    char * pUserName,
    char * pPassword,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Attach */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgatin (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    char * pPassword,
    char * pUserName,
    char * pNodeName);
/* ... */
```

API Parameters

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

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

NodeNameLen
: Input. A 2-byte unsigned integer representing the length of the node name in bytes. Set to zero if no node name is supplied.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

pPassword
: Input. A string containing the password for the specified user name. May be NULL.

pUserName
: Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

pNodeName
: Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the **DB2INSTANCE** environment variable), which can be specified as the object of an ATTACH, but cannot be used as a node name in the node directory. May be NULL.

sqleatin - Attach

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 ATTACH, 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.

Sample Programs

C	\sqllib\samples\c\dbinst.c
COBOL	\sqllib\samples\cobol\dbinst.cbl
FORTRAN	\sqllib\samples\fortran\dbinst.f
REXX	\sqllib\samples\rexx\dbinst.cmd

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 code of the application server
2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the ATTACH API)
5. Identity and platform type of the server (see the *SQL Reference*).
6. Agent ID of the agent which has been started at the server
7. Agent index
8. Node number of the server
9. Number of partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the *sqlerrmc* field of the *sqlca* (as outlined above).

If ATTACH has not been executed, instance-level APIs are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

Certain functions (**db2start**, **db2stop**, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the **DB2INSTANCE** environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated depends on the authentication type of the target instance. For detailed information about authentication types, see the *Administration Guide*.

The node to which an attachment is to be made can also be specified by a call to "sqlesetc - Set Client" on page 176 (see the SQL_ATTACH_NODE option in "SQLE-CONN-SETTING" on page 367).

See Also

"sqledtin - Detach" on page 115
"sqlesetc - Set Client" on page 176.

sqlecadb - Catalog Database

Stores database location information in the system database directory. The database can be located either on the local workstation or on a remote node.

Scope

This API affects the system database directory.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Catalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecadb (
    _SQLOLDCHAR * pDbName,
    _SQLOLDCHAR * pDbAlias,
    unsigned char Type,
    _SQLOLDCHAR * pNodeName,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    unsigned short Authentication,
    _SQLOLDCHAR * pDcePrincipal,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Catalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcadb (
    unsigned short DCEPrinLen,
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short NodeNameLen,
    unsigned short DbAliasLen,
    unsigned short DbNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pDcePrin,
    unsigned short Authentication,
    _SQLOLDCHAR * pComment,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pNodeName,
    unsigned char Type,
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pDbName);
/* ... */
```

API Parameters

DCEPrinLen
: Input. A 2-byte unsigned integer representing the length in bytes of the DCE principal. Set to zero if no principal is provided. This value should be nonzero only when authentication is specified as `SQL_AUTHENTICATION_DCE`.

CommentLen
: Input. A 2-byte unsigned integer representing the length in bytes of the comment. Set to zero if no comment is provided.

PathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the path of the local database directory. Set to zero if no path is provided.

NodeNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the node name. Set to zero if no node name is provided.

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

DbNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database name.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDcePrin
Input. A string containing the DCE principal name of the DB2 server on which the database resides. This value should only be specified when authentication is `SQL_AUTHENTICATION_DCE`. The principal must be the same as the value stored in the server's keytab file.

Authentication
Input. Contains the authentication type specified for the database. Authentication is a process that verifies that the user is who he/she claims to be. Access to database objects depends on the user's authentication. Valid values (from `sqlenv`) are:

SQL_AUTHENTICATION_SERVER
Specifies that authentication takes place on the node containing the target database.

SQL_AUTHENTICATION_CLIENT
Specifies that authentication takes place on the node where the application is invoked.

SQL_AUTHENTICATION_DCS
Specifies that authentication takes place on the node containing the target database, except when using DDCS, when it specifies that authentication takes place at the DRDA AS.

SQL_AUTHENTICATION_DCE
Specifies that authentication takes place using DCE Security Services.

SQL_AUTHENTICATION_NOT_SPECIFIED
Authentication not specified.

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.

For more information about authentication types, see the *Administration Guide*.

pComment
Input. A string containing an optional description of the database. A null string indicates no comment. The maximum length of a comment string is 30 characters.

pPath
Input. A string which, on UNIX based systems, specifies the name of the path on which the database being cataloged resides. Maximum length is 215 characters.

On OS/2 or the Windows operating system, this string specifies the letter of the drive on which the database being cataloged resides.

If a NULL pointer is provided, the default database path is assumed to be that specified by the database manager configuration parameter *dftdbpath*.

pNodeName
Input. A string containing the name of the node where the database is located. May be NULL.

Note: If neither *pPath* nor *pNodeName* is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter *dftdbpath*.

Type

Input. A single character that designates whether the database is indirect, remote, or is cataloged via DCE. Valid values (defined in `sqlenv`) are:

SQL_INDIRECT
Specifies that the database resides at this instance.

SQL_REMOTE
Specifies that the database resides at another instance.

SQL_DCE
Specifies that the database is cataloged via DCE.

pDbAlias

Input. A string containing an alias for the database.

pDbName

Input. A string containing the database name.

CATALOG DATABASE - REXX API Syntax

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]
```

REXX API Parameters

dbname

Name of the database to be cataloged.

alias

Alternate name for the database. If an alias is not specified, the database name is used as the alias.

path

Path on which the database being cataloged resides.

nodename

Name of the remote workstation where the database being cataloged resides.

Note: If neither *path* nor *nodename* is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter *dftdbpath*.

authentication

Place where authentication is to be done. Valid values are:

SERVER
Authentication occurs at the node containing the target database. This is the default.

CLIENT
Authentication occurs at the node where the application is invoked.

DCS
Specifies how authentication will take place for databases accessed using DDCS. The behavior is the same as for the type `SERVER`, except that when the authentication type is `SERVER`, DDCS forces authentication at the gateway, and when the authentication type is `DCS`, authentication is assumed to take place at the host.

DCE SERVER PRINCIPAL *dce_principal_name*
Fully qualified DCE principal name for the target server. This value is also recorded in the keytab file at the target server.

comment
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

CATALOG GLOBAL DATABASE - REXX API Syntax

```
CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [WITH comment]
```

REXX API Parameters

db_global_name
The fully qualified name that uniquely identifies the database in the DCE name space.

alias
Alternate name for the database.

DCE
The global directory service being used.

comment
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

Example

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```

Sample Programs

C	\sqllib\samples\c\dbcat.c
COBOL	\sqllib\samples\cobol\dbcat.cbl

FORTRAN	\sqllib\samples\fortran\dbcat.f
REXX	\sqllib\samples\rexx\dbcat.cmd

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`.

List the contents of the system database directory using "sqledosd - Open Database Directory Scan" on page 103, "sqledgne - Get Next Database Directory Entry" on page 100, and "sqledcls - Close Database Directory Scan" on page 98.

If directory caching is enabled (see the configuration parameter *dir_cache* in "sqlfxsys - Get Database Manager Configuration" on page 204), database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

See Also

“sqledcls - Close Database Directory Scan” on page 98
“sqledgne - Get Next Database Directory Entry” on page 100
“sqledosd - Open Database Directory Scan” on page 103
“sqleuncd - Uncatalog Database” on page 179.

sqlecran - Create Database at Node

Creates a database only on the node that calls the API. This API is not intended for general use. For example, it should be used with “sqlurst - Restore Database” on page 309 if the database partition at a node was damaged and must be recreated. Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

Note: If this API is used to recreate a database partition that was dropped (because it was damaged), the database at this node will be in the restore-pending state. After recreating the database partition, the database must immediately be restored on this node.

Scope

This API only affects the node on which it is called.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

Instance. To create a database at another node, it is necessary to first attach to that node. A database connection is temporarily established by this API during processing.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Create Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecran (
    char * pDbName,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

sqlecran - Create Database at Node

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Create Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcran (
    unsigned short reservedLen,
    unsigned short dbNameLen,
    struct sqlca * pSqlca,
    void * pReserved,
    char * pDbName);
/* ... */
```

API Parameters

reservedLen
: Input. Reserved for the length of *pReserved*.

dbNameLen
: Input. A 2-byte unsigned integer representing the length of the database name in bytes.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pReserved
: Input. A spare pointer that is set to null or points to zero. Reserved for future use.

pDbName
: Input. A string containing the name of the database to be created. Must not be NULL.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

When the database is successfully created, it is placed in restore-pending state. The database must be restored on this node before it can be used.

See Also

“sqlecrea - Create Database” on page 81
“sqledpan - Drop Database at Node” on page 106.

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 multi-node environment, this API affects all nodes that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

The node from which this API is called becomes the catalog node for the new database.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

Instance. To create a database at another (remote) node, it is necessary to first attach to that node. A database connection is temporarily established by this API during processing.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Create Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlecrea (
    char * pDbName,
    char * pLocalDbAlias,
    char * pPath,
    struct sqledbdesc * pDbDescriptor,
    struct sqledbcountryinfo * pCountryInfo,
    char Reserved2,
    void * pReserved1,
    struct sqlca * pSqlca);
/* ... */
```

sqlecrea - Create Database

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Create Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgcrea (
    unsigned short PathLen,
    unsigned short LocalDbAliasLen,
    unsigned short DbNameLen,
    struct sqlca * pSqlca,
    void * pReserved1,
    unsigned short Reserved2,
    struct sqledbcountryinfo * pCountryInfo,
    struct sqledbdesc * pDbDescriptor,
    char * pPath,
    char * pLocalDbAlias,
    char * pDbName);
/* ... */
```

API Parameters

PathLen
: Input. A 2-byte unsigned integer representing the length of the path in bytes. Set to zero if no path is provided.

LocalDbALiasLen
: Input. A 2-byte unsigned integer representing the length of the local database alias in bytes. Set to zero if no local alias is provided.

DbNameLen
: Input. A 2-byte unsigned integer representing the length of the database name in bytes.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pReserved1
: Input. A spare pointer that is set to null or points to zero.

Reserved2
: Input. Reserved for future use.

pCountryInfo
: Input. A pointer to the *sqledbcountryinfo* structure, containing the locale and the code set for the database. For more information about this structure, see “SQLEDBCOUNTRYINFO” on page 385. For a list of valid locale and code set values, see one of the *Quick Beginnings* books. May be NULL.

pDBDescriptor
: Input. A pointer to the database description block used when creating the database. The database description block may be used to supply values

that are permanently stored in the configuration file of the database, such as collating sequence. Its structure is described in “SQLEDBDESC” on page 386. May be NULL.

pPath
: Input. On UNIX based systems, specifies the path on which to create the database. If a path is not specified, the database is created on the default database path specified in the database manager configuration file (*dftdbpath* parameter). On OS/2 or the Windows operating system, specifies the letter of the drive on which to create the database. May be NULL.

Note: For MPP systems, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the *dftdbpath* database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the `$HOME` directory of the instance owner). The path specified for this API in an MPP system cannot be a relative path.

pLocalDbAlias
: Input. A string containing the alias to be placed in the client's system database directory. May be NULL. If no local alias is specified, the database name is the default.

pDbName
: Input. A string containing the database name. This is the database name that will be cataloged in the system database directory. Once the database has been successfully created in the server’s system database directory, it is automatically cataloged in the system database directory with a database alias identical to the database name. Must not be NULL.

REXX API Syntax

```
CREATE DATABASE dbname [ON path] [ALIAS dbalias]
[USING CODESET codeset TERRITORY territory]
[COLLATE USING {SYSTEM | IDENTITY | USER :udcs}]
[NUMSEGS numsegs] [DFT_EXTENT_SZ dft_extentsize]
[CATALOG TABLESPACE <tablespace_definition>]
[USER TABLESPACE <tablespace_definition>]
[TEMPORARY TABLESPACE <tablespace_definition>]
[WITH comment]

Where <tablespace_definition> stands for:
MANAGED BY {
SYSTEM USING :SMS_string |
DATABASE USING :DMS_string }
[ EXTENTSIZE number_of_pages ]
[ PREFETCHSIZE number_of_pages ]
[ OVERHEAD number_of_milliseconds ]
[ TRANSFERRATE number_of_milliseconds ]
```

REXX API Parameters

dbname
: Name of the database.

dbalias
: Alias of the database.

path
: Path on which to create the database.

  If a path is not specified, the database is created on the default database path specified in the database manager configuration file (*dftdbpath* configuration parameter).

  Note: For MPP systems, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the *dftdbpath* database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the `$HOME` directory of the instance owner). The path specified for this API in an MPP system cannot be a relative path.

codeset
: Code set to be used for data entered into the database.

territory
: Territory code (locale) to be used for data entered into the database.

SYSTEM
: Uses the collating sequence of the operating system based on the current country code.

IDENTITY
: The collating sequence is the identity sequence, where strings are compared byte for byte, starting with the leftmost byte.

USER udcs
: The collating sequence is specified by the calling application in a host variable containing a 256-byte string defining the collating sequence.

numsegs
: Number of segment directories that will be created and used to store the DAT, IDX, and LF files.

dft_extentsize
: Specifies the default *extentsize* 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 tablespace
XXX.2	Second directory name for SMS tablespace
XXX.3	and so on.

DMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, where the table space data will be stored, container sizes (specified in a number of 4KB pages) and types (file or device). The specified devices (not files) must already exist. In the following, XXX represents the host variable name. Note that each of the container names cannot exceed 254 bytes in length.

XXX.0	Number of strings in the REXX host variable (number of first level elements)
XXX.1.1	Type of the first container (`file` or `device`)
XXX.1.2	First file name or device name
XXX.1.3	Size (in pages) of the first container
XXX.2.1	Type of the second container (`file` or `device`)
XXX.2.2	Second file name or device name
XXX.2.3	Size (in pages) of the second container
XXX.3.1	and so on.

EXTENTSIZE number_of_pages
Number of 4KB pages that will be written to a container before skipping to the next container.

PREFETCHSIZE number_of_pages
Number of 4KB pages that will be read from the table space when data prefetching is being performed.

OVERHEAD number_of_milliseconds
Number that specifies the I/O controller overhead, disk seek, and latency time in milliseconds.

TRANSFERRATE number_of_milliseconds
Number that specifies the time in milliseconds to read one 4KB page into memory.

comment
Description of the database or the database entry in the system directory. Do not use a carriage return or line feed character in the comment. Be sure to enclose the comment text in double quotation marks. Maximum size is 30 characters.

Sample Programs

C	\sqllib\samples\c\dbconf.c
COBOL	\sqllib\samples\cobol\dbconf.cbl
FORTRAN	\sqllib\samples\fortran\dbconf.f
REXX	\sqllib\samples\rexx\dbconf.cmd

Usage Notes

CREATE DATABASE:

- Creates a database in the specified subdirectory. In an MPP system, creates the database on all nodes listed in `db2nodes.cfg`, and creates a `$DB2INSTANCE/NODE`*xxxx* directory under the specified subdirectory at each node, where *xxxx* represents the local node number. In a non-MPP system, creates a `$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 server node on which this API is called becomes the catalog node for the new database. Two nodegroups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP. For more information, see the *SQL Reference*.
- Binds the previously defined database manager bind files to the database (these are listed in `db2ubind.lst`). If one or more of these files do not bind successfully, **sqlecrea** returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC.
- Creates SYSCATSPACE, TEMPSPACE1, and USERSPACE1 table spaces. The SYSCATSPACE table space is only created on the catalog node. All nodes have the same table space definitions.
- Grants the following:
  - DBADM authority, and CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, and IMPLICIT_SCHEMA privileges to the database creator

- CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA privileges to PUBLIC
- SELECT privilege on each system catalog to PUBLIC
- BIND and EXECUTE privilege to PUBLIC for each successfully bound utility.

With *dbadm* authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with *sysadm* or *dbadm* authority over the database revokes these privileges, the database creator nevertheless retains them.

In an MPP environment, the database manager creates a subdirectory, `$DB2INSTANCE/NODE`*xxxx*, under the specified or default path on all nodes. The *xxxx* is the node number as defined in the `db2nodes.cfg` file (that is, node 0 becomes `NODE0000`). Subdirectories `SQL00001` through `SQL`*nnnnn* will reside on this path. This ensures that the database objects associated with different nodes are stored in different directories (even if the subdirectory `$DB2INSTANCE` under the specified or default path is shared by all nodes).

CREATE DATABASE will fail if the application is already connected to a database.

If the database description block structure is not set correctly, an error message is returned (see “SQLEDBDESC” on page 386).

The "eye-catcher" of the database description block must be set to the symbolic value `SQLE_DBDESC_2` (defined in `sqlenv`). The following sample user-defined collating sequences are available in the host language include files:

sqle819a If the code page of the database is 819 (ISO Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).

sqle819b If the code page of the database is 819 (ISO Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

sqle850a If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).

sqle850b If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

sqle932a If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5035 (EBCDIC Japanese).

sqle932b If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5026 (EBCDIC Japanese).

The collating sequence specified during CREATE DATABASE cannot be changed later, and all character comparisons in the database use the specified collating sequence. This affects the structure of indexes as well as the results of queries.

Use **sqlecadb** to define different alias names for the new database.

See Also

"sqlabndx - Bind" on page 10
"sqlecadb - Catalog Database" on page 72
"sqlecran - Create Database at Node" on page 79
"sqledpan - Drop Database at Node" on page 106
"sqledrpd - Drop Database" on page 110.

sqlectnd - Catalog Node

Stores information in the node directory about the location of a DB2 server instance based on the communications protocol used to access that instance. The information is needed to establish a database connection or attachment between an application and a server instance.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Catalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlectnd (
    struct sqle_node_struct * pNodeInfo,
    void * pProtocolInfo,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Catalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgctnd (
    struct sqlca * pSqlca,
    struct sqle_node_struct * pNodeInfo,
    void * pProtocolInfo);
/* ... */
```

API Parameters

pNodeInfo
: Input. A pointer to a node directory structure. For more information about this structure, see “SQLE-NODE-STRUCT” on page 377.

pProtocolInfo
: Input. A pointer to the protocol structure. For more information about these structures, see:
- “SQLE-NODE-CPIC” on page 372
- “SQLE-NODE-IPXSPX” on page 373
- “SQLE-NODE-LOCAL” on page 374
- “SQLE-NODE-NETB” on page 375
- “SQLE-NODE-NPIPE” on page 376
- “SQLE-NODE-TCPIP” on page 379.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

CATALOG APPC NODE - REXX API Syntax

```
CATALOG APPC NODE nodename DESTINATION symbolic_destination_name
[SECURITY {NONE|SAME|PROGRAM}]
[WITH comment]
```

REXX API Parameters

nodename
: Alias for the node to be cataloged.

symbolic_destination_name
: Symbolic destination name of the remote partner node.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

CATALOG IPX/SPX NODE - REXX API Syntax

```
CATALOG IPXSPX NODE nodename REMOTE file_server SERVER objectname
[WITH comment]
```

REXX API Parameters

nodename
: Alias for the node to be cataloged.

file_server
: Name of the NetWare file server where the internetwork address of the database manager instance is registered. The internetwork address is stored in the bindery at the NetWare file server, and is accessed using *objectname*.

objectname
: The database manager server instance is represented as the object, *objectname*, on the NetWare file server. The server's IPX/SPX internetwork address is stored and retrieved from this object.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

CATALOG LOCAL NODE - REXX API Syntax

```
CATALOG LOCAL NODE nodename INSTANCE instance_name [WITH comment]
```

REXX API Parameters

nodename
: Alias for the node to be cataloged.

instance_name
: Name of the instance to be cataloged.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

CATALOG NETBIOS NODE - REXX API Syntax

```
CATALOG NETBIOS NODE nodename REMOTE server_nname ADAPTER adapternum
[WITH comment]
```

REXX API Parameters

nodename
: Alias for the node to be cataloged.

server_nname
: Name of the remote workstation. This is the workstation name (*nname*) found in the database manager configuration file of the server instance.

adapternum
: Local LAN adapter number.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

CATALOG NPIPE NODE - REXX API Syntax

```
CATALOG NPIPE NODE nodename REMOTE computer_name INSTANCE instance_name
```

REXX API Parameters

nodename
: Alias for the node to be cataloged.

computer_name
: The computer name of the node on which the target database resides.

instance_name
: Name of the instance to be cataloged.

CATALOG TCPIP NODE - REXX API Syntax

```
CATALOG TCPIP NODE nodename REMOTE hostname SERVER servicename
[WITH comment]
```

Parameters

nodename
: Alias for the node to be cataloged.

hostname
: Host name of the node where the target database resides.

servicename
: Either the service name of the database manager instance on the remote node, or the port number associated with that service name.

comment
: An optional description associated with this node directory entry. Do not include a CR/LF character in a comment. Maximum length is 30 characters. The comment text must be enclosed by double quotation marks.

Sample Programs

C	\sqllib\samples\c\nodecat.c
COBOL	\sqllib\samples\cobol\nodecat.cbl
FORTRAN	\sqllib\samples\fortran\nodecat.f
REXX	\sqllib\samples\rexx\nodecat.cmd

Usage Notes

DB2 creates the node directory on the first call to this API if the node directory does not exist. On OS/2 or the Windows operating system, the node directory is stored in the directory of the instance being used. On UNIX based systems, it is stored in the DB2 install directory (`sqllib`, for example).

If directory caching is enabled (see the configuration parameter *dir_cache* in “sqlfxsys - Get Database Manager Configuration” on page 204), database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

To list the contents of the node directory, use “sqlenops - Open Node Directory Scan” on page 153, “sqlengne - Get Next Node Directory Entry” on page 150, and “sqlencls - Close Node Directory Scan” on page 148.

See Also

“sqlencls - Close Node Directory Scan” on page 148
“sqlengne - Get Next Node Directory Entry” on page 150
“sqlenops - Open Node Directory Scan” on page 153
“sqleuncn - Uncatalog Node” on page 182.

sqledcgd - Change Database Comment

Changes a database comment in the system database directory or the local database directory. New comment text can be substituted for text currently associated with a comment.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Change Database Comment */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcgd (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pComment,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Change Database Comment */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcgd (
    unsigned short CommentLen,
    unsigned short PathLen,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pComment,
    _SQLOLDCHAR * pPath,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API Parameters

CommentLen
: Input. A 2-byte unsigned integer representing the length in bytes of the comment. Set to zero if no comment is provided.

PathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the path parameter. Set to zero if no path is provided.

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pComment
: Input. A string containing an optional description of the database. A null string indicates no comment. It can also indicate no change to an existing database comment.

pPath
: Input. A string containing the path on which the local database directory resides. If the specified path is a null pointer, the system database directory is used.

The comment is only changed in the local database directory or the system database directory on the node on which the API is executed. To change the database comment on all nodes, run the API on every node.

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.

sqledcgd - Change Database Comment

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.

Sample Programs

C	\sqllib\samples\c\dbcmt.c
COBOL	\sqllib\samples\cobol\dbcmt.cbl
FORTRAN	\sqllib\samples\fortran\dbcmt.f
REXX	\sqllib\samples\rexx\dbcmt.cmd

Usage Notes

New comment text replaces existing text. To append information, enter the old comment text, followed by the new text.

To modify an existing comment:

1. Call “sqledosd - Open Database Directory Scan” on page 103
2. Call “sqledgne - Get Next Database Directory Entry” on page 100 to retrieve the old comment
3. Modify the retrieved comment
4. Call “sqledcls - Close Database Directory Scan” on page 98
5. Call "sqledcgd - Change Database Comment" to replace the old text with the modified 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.

See Also

"sqlecrea - Create Database" on page 81
"sqlecadb - Catalog Database" on page 72.

sqledcls - Close Database Directory Scan

Frees the resources allocated by “sqledosd - Open Database Directory Scan” on page 103.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Close Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqledcls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Close Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdcls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Handle

Input. Identifier returned from the associated OPEN DATABASE DIRECTORY SCAN API.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
CLOSE DATABASE DIRECTORY scanid
```

REXX API Parameters

scanid
: A host variable containing the *scanid* returned from the OPEN DATABASE DIRECTORY SCAN API.

Sample Programs

C	\sqllib\samples\c\dbcat.c
COBOL	\sqllib\samples\cobol\dbcat.cbl
FORTRAN	\sqllib\samples\fortran\dbcat.f
REXX	\sqllib\samples\rexx\dbcat.cmd

See Also

“sqledgne - Get Next Database Directory Entry” on page 100
“sqledosd - Open Database Directory Scan” on page 103.

sqledgne - Get Next Database Directory Entry

Returns the next entry in the system database directory or the local database directory copy returned by "sqledosd - Open Database Directory Scan" on page 103. Subsequent calls to this API return additional entries.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Get Next Database Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqledgne (
    unsigned short Handle,
    struct sqledinfo ** ppDbDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Get Next Database Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdgne (
    unsigned short Handle,
    struct sqledinfo ** ppDbDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Handle
: Input. Identifier returned from the associated OPEN DATABASE DIRECTORY SCAN API.

ppDbDirEntry
: Output. The caller supplies the API with the address of a pointer to an *sqledinfo* structure. The space for the directory data is allocated by the API, and a pointer to that space is returned to the caller. A call to "sqledcls - Close Database Directory Scan" on page 98 frees the allocated space. Information returned to the buffer is described in "SQLEDINFO" on page 394.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

```
GET DATABASE DIRECTORY ENTRY :scanid [USING :value]
```

REXX API Parameters

scanid
: A REXX host variable containing the identifier returned from the OPEN DATABASE DIRECTORY SCAN API.

value
: A compound REXX host variable to which the database entry information is returned. If no name is given, the name `SQLDINFO` is used. In the following, XXX represents the host variable name (the corresponding field names are taken from the structure returned by the API):

XXX.0	Number of elements in the variable (always 12)
XXX.1	ALIAS (alias of the database)
XXX.2	DBNAME (name of the database)
XXX.3	DRIVE/PATH (local database directory path name)
XXX.3.1	NODE NUMBER (valid for local database directory only)
XXX.4	INTNAME (token identifying the database subdirectory)
XXX.5	NODENAME (name of the node where the database is located)
XXX.6	DBTYPE (product name and release number)
XXX.7	COMMENT (comment associated with the database)
XXX.8	Reserved
XXX.9	TYPE (entry type)

XXX.10	AUTHENTICATION (authentication type)
XXX.10.1	DCE principal
XXX.11	GLBDBNAME (Global database name)
XXX.12	CATALOG NODE NUMBER

Sample Programs

C	\sqllib\samples\c\dbcat.c
COBOL	\sqllib\samples\cobol\dbcat.cbl
FORTRAN	\sqllib\samples\fortran\dbcat.f
REXX	\sqllib\samples\rexx\dbcat.cmd

Usage Notes

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

A subsequent GET NEXT DATABASE DIRECTORY ENTRY obtains the entry following the current entry.

The *sqlcode* value of *sqlca* is set to 1014 if there are no more entries to scan when GET NEXT DATABASE DIRECTORY ENTRY is called.

The count value returned by the OPEN DATABASE DIRECTORY SCAN API can be used to scan through the entire directory by issuing GET NEXT DATABASE DIRECTORY ENTRY calls, one at a time, until the number of scans equals the count of entries.

See Also

“sqledcls - Close Database Directory Scan” on page 98
“sqledosd - Open Database Directory Scan” on page 103.

sqledosd - Open Database Directory Scan

Stores a copy of the system database directory or the local database directory in memory, and returns the number of entries. This copy represents a snapshot of the directory at the time the directory is opened. This copy is not updated, even if the directory itself is changed later.

Use “sqledgne - Get Next Database Directory Entry” on page 100 to advance through the database directory, examining information about the database entries. Close the scan using “sqledcls - Close Database Directory Scan” on page 98. This removes the copy of the directory from memory.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Open Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqledosd (
    _SQLOLDCHAR * pPath,
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

sqledosd - Open Database Directory Scan

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Open Database Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdosd (
    unsigned short PathLen,
    struct sqlca * pSqlca,
    unsigned short * pNumEntries,
    unsigned short * pHandle,
    _SQLOLDCHAR * pPath);
/* ... */
```

API Parameters

PathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the path parameter. Set to zero if no path is provided.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pNumEntries
: Output. Address of a 2-byte area where the number of directory entries is returned.

pHandle
: Output. Address of a 2-byte area for the returned identifier. This identifier must be passed to “sqledgne - Get Next Database Directory Entry” on page 100 for scanning the database entries, and to “sqledcls - Close Database Directory Scan” on page 98 to release the resources.

pPath
: Input. The name of the path on which the local database directory resides. If the specified path is a NULL pointer, the system database directory is used.

REXX API Syntax

```
OPEN DATABASE DIRECTORY [ON path_name] USING :value
```

REXX API Parameters

path_name
: Name of the path on which the local database directory resides. If the path is not specified, the system database directory is used.

value

A compound REXX host variable to which database directory information is returned. In the following, XXX represents the host variable name.

XXX.0	Number of elements in the variable (always 2)
XXX.1	Identifier (handle) for future scan access
XXX.2	Number of entries contained within the directory.

Sample Programs

C	\sqllib\samples\c\dbcat.c
COBOL	\sqllib\samples\cobol\dbcat.cbl
FORTRAN	\sqllib\samples\fortran\dbcat.f
REXX	\sqllib\samples\rexx\dbcat.cmd

Usage Notes

Storage allocated by this API is freed by “sqledcls - Close Database Directory Scan” on page 98.

Multiple OPEN DATABASE DIRECTORY SCAN APIs can be issued against the same directory. However, the results may not be the same. The directory may change between openings.

There can be a maximum of eight opened database directory scans per process.

See Also

“sqledcls - Close Database Directory Scan” on page 98
“sqledgne - Get Next Database Directory Entry” on page 100.

sqledpan - Drop Database at Node

Drops a database at a specified node. Can only be run on an MPP server.

Scope

This API only affects the node on which it is called.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None. An instance attachment is established for the duration of the call.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Drop Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqledpan (
    char * pDbAlias,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Drop Database at Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdpan (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    void * pReserved2,
    char * pDbAlias);
/* ... */
```

API Parameters

Reserved1
: Reserved for future use.

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pReserved2
: A spare pointer that is set to null or points to zero. Reserved for future use.

pDbAlias
: Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

This API is used by utilities supplied with DB2 Universal Database Extended Enterprise Edition, and is not intended for general use. Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

See Also

“sqlecran - Create Database at Node” on page 79
“sqledrpd - Drop Database” on page 110.

sqledreg - Deregister

Deregisters the DB2 server from a network file server. The DB2 server's network address is removed from a specified registry on the file server.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Deregister */
/* ... */
SQL_API_RC SQL_API_FN
  sqledreg (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Deregister */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdreg (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Registry

Input. Indicates where on the network file server to deregister the DB2 server. In this release, the only supported registry is `SQL_NWBINDERY` (NetWare file server bindery, defined in `sqlenv`).

pRegisterInfo
: Input. A pointer to the *sqle_reg_nwbindery* structure. In this structure, the caller specifies a user name and password that are valid on the network file server. For more information about this structure, see "SQLE-REG-NWBINDERY" on page 380.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\regder.c
COBOL	\sqllib\samples\cobol\regder.cbl
FORTRAN	\sqllib\samples\fortran\regder.f

Usage Notes

When *Registry* has a value of `SQL_NWBINDERY`, this API uses the NetWare user name and password supplied in the *sqle_reg_nwbindery* structure to log onto the NetWare file server (FILESERVER) specified in the database manager configuration file. The object name (OBJECTNAME) specified in the database manager configuration file is deleted from the NetWare file server bindery.

The NetWare user name and password specified must have supervisory or equivalent authority.

This API *must* be issued locally from the DB2 server. It is not supported remotely.

If the IPX/SPX fields are reconfigured, or the DB2 server's IPX/SPX internetwork address changes, deregister the DB2 server from the network file server before making the changes, and then register it again after the changes have been made.

See Also

"sqleregs - Register" on page 165.

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 nodes that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

Instance. It is not necessary to call ATTACH before dropping a remote database. If the database is cataloged as remote, an instance attachment to the remote node is established for the duration of the call.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Drop Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpd (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pReserved2,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Drop Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpd (
    unsigned short Reserved1,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pReserved2,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API Parameters

Reserved1
: Reserved for future use.

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pReserved2
: A spare pointer that is set to null or points to zero. Reserved for future use.

pDbAlias
: Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

REXX API Syntax

```
DROP DATABASE dbalias
```

REXX API Parameters

dbalias
: The alias of the database to be dropped.

Sample Programs

C	\sqllib\samples\c\dbconf.sqc
COBOL	\sqllib\samples\cobol\dbconf.sqb

FORTRAN	\sqllib\samples\fortran\dbconf.sqf
REXX	\sqllib\samples\rexx\dbconf.cmd

Usage Notes

DROP DATABASE deletes all user data and log files. If the log files are needed for a roll-forward recovery after a restore operation, the files should be saved prior to calling this API.

The database must not be in use; all users must be disconnected from the database before the database can be dropped.

To be dropped, a database must be cataloged in the system database directory. Only the specified database alias is removed from the system database directory. If other aliases with the same database name exist, their entries remain. If the database being dropped is the last entry in the local database directory, the local database directory is deleted automatically.

If DROP DATABASE is issued 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.

See Also

"sqlecadb - Catalog Database" on page 72
"sqlecrea - Create Database" on page 81
"sqlecran - Create Database at Node" on page 79
"sqledpan - Drop Database at Node" on page 106
"sqleuncd - Uncatalog Database" on page 179.

sqledrpn - Drop Node Verify

Verifies whether a node is being used by a database. A message is returned, indicating whether the node can be dropped.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
sysctrl

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Drop Node Verify */
/* ... */
SQL_API_RC SQL_API_FN
  sqledrpn (
    unsigned short Action,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Drop Node Verify */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrpn (
    unsigned short Reserved1,
    struct sqlca * pSqlca,
    void * pReserved2,
    unsigned short Action);
/* ... */
```

sqledrpn - Drop Node Verify

API Parameters

Reserved1
: Reserved for the length of *pReserved2*.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pReserved2
: A spare pointer that is set to NULL or points to 0. Reserved for future use.

Action
: The action requested. The valid value is:

  SQL_DROPNODE_VERIFY

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

If a message is returned, indicating that the node is not in use, use the **db2stop** command with DROP NODENUM to remove the entry for the node from the `db2nodes.cfg` file, which removes the node from the database system.

If a message is returned, indicating that the node is in use, the following actions should be taken:

1. If the node contains data, redistribute the data to remove it from the node using “sqludrdt - Redistribute Nodegroup” on page 237. Use either the drop node option on the **sqludrdt** API, or the ALTER NODEGROUP statement to remove the node from any nodegroups for the database. This must be done for each database that contains the node in a nodegroup. For more information, see the *SQL Reference*.
2. Drop any event monitors that are defined on the node.
3. Rerun **sqledrpn** to ensure that the database is no longer in use.

See Also

“sqleaddn - Add Node” on page 65
“sqlepstp - Stop Database Manager” on page 159.

sqledtin - Detach

Removes the logical instance attachment, and terminates the physical communication connection if there are no other logical connections using this layer.

Authorization

None

Required Connection

None. Removes an existing instance attachment.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Detach */
/* ... */
SQL_API_RC SQL_API_FN
  sqledtin (
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Detach */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdtin (
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqledtin - Detach

REXX API Syntax

```
DETACH
```

Sample Programs

C	\sqllib\samples\c\dbinst.c
COBOL	\sqllib\samples\cobol\dbinst.cbl
FORTRAN	\sqllib\samples\fortran\dbinst.f
REXX	\sqllib\samples\rexx\dbinst.cmd

See Also

"sqleatin - Attach" on page 68.

sqlefmem - Free Memory

Frees memory allocated by DB2 APIs on the caller's behalf. Intended for use with “sqlbtcq - Tablespace Container Query” on page 54 and “sqlbmtsq - Tablespace Query” on page 39.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Free Memory */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefmem (
    struct sqlca * pSqlca,
    void * pBuffer);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Free Memory */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfmem (
    struct sqlca * pSqlca,
    void * pBuffer);
/* ... */
```

API Parameters

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pBuffer
Input. Pointer to the memory to be freed.

Sample Programs

C	\sqllib\samples\c\tabspace.sqc
COBOL	\sqllib\samples\cobol\tspace.sqb
FORTRAN	\sqllib\samples\fortran\tspace.sqf

sqlefrce - Force Application

Forces local or remote users or applications off the system to allow for maintenance on a server.

Attention: If an operation that cannot be interrupted (RESTORE DATABASE, for example) is forced, the operation must be successfully re-executed before the database becomes available.

Scope

This API affects all nodes that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

Instance. To force users off a remote server, it is necessary to first attach to that server. If no attachment exists, this API is executed locally.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Force Application */
/* ... */
SQL_API_RC SQL_API_FN
  sqlefrce (
    long NumAgentIds,
    unsigned long * pAgentIds,
    unsigned short ForceMode,
    struct sqlca * pSqlca);
/* ... */
```

sqlefrce - Force Application

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Force Application */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgfrce (
    struct sqlca * pSqlca,
    unsigned short ForceMode,
    unsigned long * pAgentIds,
    long NumAgentIds);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

ForceMode
: Input. An integer specifying the operating mode of the FORCE APPLICATION API. Only the asynchronous mode is supported. This means that FORCE APPLICATION does not wait until all specified users are terminated before returning. It returns as soon as the API has been issued successfully, or an error occurs. As a result, there may be a short interval between the time the FORCE APPLICATION call completes and the specified users have been terminated.

  This parameter must be set to `SQL_ASYNCH` (defined in `sqlenv`).

pAgentIds
: Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding database user. To list the agent IDs of the active applications, use “sqlmonss - Get Snapshot” on page 215.

NumAgentIds
: Input. An integer representing the total number of users to be terminated. This number should be the same as the number of elements in the array of agent IDs.

  If this parameter is set to `SQL_ALL_USERS` (defined in `sqlenv`), all users are forced. If it is set to zero, an error is returned.

REXX API Syntax

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API Parameters

ALL
: All applications will be disconnected from their database connection.

agentidarray
: A compound REXX host variable containing the list of agent IDs to be terminated. In the following, XXX is the name of the host variable:

XXX.0	Number of agents to be terminated
XXX.1	First agent ID
XXX.2	Second agent ID
XXX.3	and so on.

ASYNC
: The only mode currently supported means that FORCE APPLICATION does not wait until all specified applications are terminated before returning.

Sample Programs

C	\sqllib\samples\c\dbstop.sqc
COBOL	\sqllib\samples\cobol\dbstop.sqb
FORTRAN	\sqllib\samples\fortran\dbstop.sqf
REXX	\sqllib\samples\rexx\dbstop.cmd

Usage Notes

db2stop cannot be executed during a force. The database manager remains active so that subsequent database manager operations can be handled without the need for **db2start**.

To preserve database integrity, only users who are idling or executing interruptible database operations can be terminated.

After a FORCE has been issued, the database will still accept requests to connect. Additional forces may be required to completely force all users off.

The database system monitor functions are used to gather the agent IDs of the users to be forced. For more information, see the *System Monitor Guide and Reference*.

When the force mode is set to `SQL_ASYNCH` (the only value permitted), the API immediately returns to the calling application.

Minimal validation is performed on the array of agent IDs to be forced. The user must ensure that the pointer points to an array containing the total number of elements specified. If *NumAgentIds* is set to `SQL_ALL_USERS`, the array is ignored.

When a user is terminated, a ROLLBACK is performed to ensure database consistency.

All users that can be forced will be forced. If one or more specified agent IDs cannot be found, *sqlcode* in the *sqlca* structure is set to 1230. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and **sqlefrce** is called. The user that calls this API is never forced off.

Agent IDs are recycled, and are used to force applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the wrong user may be forced.

See Also

"sqleatin - Attach" on page 68
"sqledtin - Detach" on page 115
"sqlepstp - Stop Database Manager" on page 159
"sqlmonss - Get Snapshot" on page 215.

sqlegdad - Catalog DCS Database

Stores information about a remote database in the Database Connection Services (DCS) directory. Such databases are accessed through an Application Requester (AR), such as Distributed Database Connection Services (DDCS). 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. For more information about DDCS and DCS directory entries, see the *DB2 Connect User's Guide*.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Catalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdad (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Catalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdad (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDCSDirEntry
: Input. A pointer to an *sql_dir_entry* (Database Connection Services directory) structure. For more information about this structure, see “SQL-DIR-ENTRY” on page 343.

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.

Sample Programs

C	\sqllib\samples\c\dcscat.c
COBOL	\sqllib\samples\cobol\dcscat.cbl
FORTRAN	\sqllib\samples\fortran\dcscat.f
REXX	\sqllib\samples\rexx\dcscat.cmd

Usage Notes

The DB2 Connect program provides connections to DRDA Application Servers such as:

- DATABASE 2 (DB2) for MVS databases on System/370 and System/390 architecture host computers
- Structured Query Language/Data System (SQL/DS) databases on System/370 and System/390 architecture host computers
- OS/400 databases on Application System/400 (AS/400) host computers.

The database manager creates a Database Connection Services directory if one does not exist. This directory is stored on the path that contains the database manager instance that is being used. The DCS directory is maintained outside of the database.

The database must also be cataloged as a remote database in the system database directory.

List the contents of the DCS directory using “sqlegdsc - Open DCS Directory Scan” on page 136, “sqlegdge - Get DCS Directory Entry for Database” on page 131, “sqlegdgt - Get DCS Directory Entries” on page 133, and “sqlegdcl - Close DCS Directory Scan” on page 126.

Note: If directory caching is enabled (see the configuration parameter *dir_cache* in “sqlfxsys - Get Database Manager Configuration” on page 204), database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

See Also

“sqlegdel - Uncatalog DCS Database” on page 128.

sqlegdcl - Close DCS Directory Scan

Frees the resources that are allocated by “sqlegdsc - Open DCS Directory Scan” on page 136.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Close DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdcl (
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Close DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdcl (
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
CLOSE DCS DIRECTORY
```

Sample Programs

C	\sqllib\samples\c\dcscat.c
COBOL	\sqllib\samples\cobol\dcscat.cbl
FORTRAN	\sqllib\samples\fortran\dcscat.f
REXX	\sqllib\samples\rexx\dcscat.cmd

See Also

"sqlegdgt - Get DCS Directory Entries" on page 133
"sqlegdsc - Open DCS Directory Scan" on page 136.

sqlegdel - Uncatalog DCS Database

Deletes an entry from the Database Connection Services (DCS) directory.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Uncatalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdel (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Uncatalog DCS Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdel (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDCSDirEntry
: Input/Output. A pointer to the Database Connection Services directory structure. For more information about this structure, see "SQL-DIR-ENTRY" on page 343. Fill in the *ldb* field of this structure with the local name of the database to be deleted. The DCS directory entry with a matching local database name is copied to this structure before being deleted.

REXX API Syntax

```
UNCATALOG DCS DATABASE dbname [USING :value]
```

REXX API Parameters

dbname
: The local database name of the directory entry to be deleted.

value
: A compound REXX host variable into which the directory entry information is returned. In the following, XXX represents the host variable name. If no name is given, the name `SQLGWINF` is used.

XXX.0	Number of elements in the variable (always 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED.

Sample Programs

C	\sqllib\samples\c\dcscat.c
COBOL	\sqllib\samples\cobol\dcscat.cbl
FORTRAN	\sqllib\samples\fortran\dcscat.f
REXX	\sqllib\samples\rexx\dcscat.cmd

Usage Notes

DCS databases are also cataloged in the system database directory as remote databases that can be uncataloged using "sqleuncd - Uncatalog Database" on page 179.

To recatalog a database in the DCS directory, use "sqlegdad - Catalog DCS Database" on page 123.

To list the DCS databases that are cataloged on a node, use “sqlegdsc - Open DCS Directory Scan” on page 136, “sqlegdgt - Get DCS Directory Entries” on page 133, and “sqlegdcl - Close DCS Directory Scan” on page 126.

If directory caching is enabled (see the configuration parameter *dir_cache* in “sqlfxsys - Get Database Manager Configuration” on page 204), database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

See Also

“sqlegdad - Catalog DCS Database” on page 123
“sqlegdcl - Close DCS Directory Scan” on page 126
“sqlegdge - Get DCS Directory Entry for Database” on page 131
“sqlegdgt - Get DCS Directory Entries” on page 133
“sqlegdsc - Open DCS Directory Scan” on page 136
“sqleuncd - Uncatalog Database” on page 179.

sqlegdge - Get DCS Directory Entry for Database

Returns information for a specific entry in the Database Connection Services (DCS) directory.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entry for Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdge (
    struct sql_dir_entry * pDCSDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entry for Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdge (
    struct sqlca * pSqlca,
    struct sql_dir_entry * pDCSDirEntry);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDCSDirEntry
: Input/Output. Pointer to the Database Connection Services directory structure. For more information about this structure, see “SQL-DIR-ENTRY” on page 343. Fill in the *ldb* field of this structure with the local name of the database whose DCS directory entry is to be retrieved. The remaining fields in the structure are filled in upon return of this API.

REXX API Syntax

```
GET DCS DIRECTORY ENTRY FOR DATABASE dbname [USING :value]
```

REXX API Parameters

dbname
: Specifies the local database name of the directory entry to be obtained.

value
: A compound REXX host variable into which the directory entry information is returned. In the following, XXX represents the host variable name. If no name is given, the name `SQLGWINF` is used.

XXX.0	Number of elements in the variable (always 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED.

Sample Programs

C	\sqllib\samples\c\dcscat.c
COBOL	\sqllib\samples\cobol\dcscat.cbl
FORTRAN	\sqllib\samples\fortran\dcscat.f
REXX	\sqllib\samples\rexx\dcscat.cmd

See Also

“sqlegdad - Catalog DCS Database” on page 123
“sqlegdcl - Close DCS Directory Scan” on page 126
“sqlegdel - Uncatalog DCS Database” on page 128
“sqlegdgt - Get DCS Directory Entries” on page 133
“sqlegdsc - Open DCS Directory Scan” on page 136.

sqlegdgt - Get DCS Directory Entries

Transfers a copy of Database Connection Services (DCS) directory entries to a buffer supplied by the application.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entries */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdgt (
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Get DCS Directory Entries */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdgt (
    struct sqlca * pSqlca,
    short * pNumEntries,
    struct sql_dir_entry * pDCSDirEntries);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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. For more information about this structure, see "SQL-DIR-ENTRY" on page 343. The buffer must be large enough to hold the number of entries specified in the *pNumEntries* parameter.

REXX API Syntax

```
GET DCS DIRECTORY ENTRY [USING :value]
```

REXX API Parameters

value
: A compound REXX host variable into which the directory entry information is returned. In the following, XXX represents the host variable name. If no name is given, the name `SQLGWINF` is used.

XXX.0	Number of elements in the variable (always 7)
XXX.1	RELEASE
XXX.2	LDB
XXX.3	TDB
XXX.4	AR
XXX.5	PARMS
XXX.6	COMMENT
XXX.7	RESERVED.

Sample Programs

C	\sqllib\samples\c\dcscat.c
COBOL	\sqllib\samples\cobol\dcscat.cbl
FORTRAN	\sqllib\samples\fortran\dcscat.f
REXX	\sqllib\samples\rexx\dcscat.cmd

Usage Notes

"sqlegdsc - Open DCS Directory Scan" on page 136, which returns the entry count, must be called prior to issuing GET DCS DIRECTORY ENTRIES.

If all entries are copied to the caller, the Database Connection Services directory scan is automatically closed, and all resources are released.

If entries remain, subsequent calls to this API should be made, or CLOSE DCS DIRECTORY SCAN should be called, to release system resources.

See Also

"sqlegdcl - Close DCS Directory Scan" on page 126
"sqlegdge - Get DCS Directory Entry for Database" on page 131
"sqlegdsc - Open DCS Directory Scan" on page 136.

sqlegdsc - Open DCS Directory Scan

Stores a copy in memory of the Database Connection Services directory entries, and returns the number of entries. This is a snapshot of the directory at the time the directory is opened.

The copy is not updated if the directory itself changes after a call to this API. Use "sqlegdgt - Get DCS Directory Entries" on page 133 to retrieve the entries, and "sqlegdcl - Close DCS Directory Scan" on page 126 to release the resources associated with calling this API.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Open DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegdsc (
    short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Open DCS Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggdsc (
    struct sqlca * pSqlca,
    short * pNumEntries);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pNumEntries
: Output. Address of a 2-byte area to which the number of directory entries is returned.

REXX API Syntax

```
OPEN DCS DIRECTORY
```

Sample Programs

C	\sqllib\samples\c\dcscat.c
COBOL	\sqllib\samples\cobol\dcscat.cbl
FORTRAN	\sqllib\samples\fortran\dcscat.f
REXX	\sqllib\samples\rexx\dcscat.cmd

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.

See Also

“sqlegdcl - Close DCS Directory Scan” on page 126
“sqlegdge - Get DCS Directory Entry for Database” on page 131
“sqlegdgt - Get DCS Directory Entries” on page 133.

sqlegins - Get Instance

Returns the value of the **DB2INSTANCE** environment variable.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Get Instance */
/* ... */
SQL_API_RC SQL_API_FN
  sqlegins (
    _SQLOLDCHAR * pInstance,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Get Instance */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggins (
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pInstance);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pInstance
: Output. Pointer to a string buffer where the database manager instance name is placed. This buffer must be at least 8 bytes in length.

REXX API Syntax

```
GET INSTANCE INTO :instance
```

REXX API Parameters

instance
A REXX host variable into which the database manager instance name is to be placed.

Sample Programs

C	\sqllib\samples\c\dbinst.c
COBOL	\sqllib\samples\cobol\dbinst.cbl
FORTRAN	\sqllib\samples\fortran\dbinst.f
REXX	\sqllib\samples\rexx\dbinst.cmd

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” on page 68, with null arguments except for the *sqlca* structure.

sqleintr - Interrupt

Stops a request. This API is called from a control break signal handler in an application. The control break signal handler can be the default, installed by “sqleisig - Install Signal Handler” on page 143, or a routine supplied by the programmer and installed using an appropriate operating system call.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Interrupt */
/* ... */
SQL_API_RC SQL_API_FN
  sqleintr (
    void);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Interrupt */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgintr (
    void);
/* ... */
```

API Parameters

The INTERRUPT API does not accept any parameters.

REXX API Syntax

```
INTERRUPT
```

Example

```
call SQLDBS 'INTERRUPT'
```

Usage Notes

No database manager APIs should be called from an interrupt handler except the INTERRUPT API. 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 on other APIs:

Table 5. INTERRUPT Actions

Database Activity	**Action**
IMPORT/EXPORT	Utility cancelled. Database updates rolled back.
REORGANIZE TABLE	Utility cancelled. Table is left in its previous state.
BACKUP	Utility cancelled. Data on media may be incomplete.
RESTORE	Utility cancelled. DROP DATABASE performed. Not applicable to table space level restore.
LOAD	Utility cancelled. Data in table may be incomplete.
PREP	Precompile cancelled. Package creation rolled back.
BIND	Binding cancelled. Package creation rolled back.
COMMIT	None. COMMIT completes.
FORCE APPLICATION	None. FORCE APPLICATION completes.
ROLLBACK	None. ROLLBACK completes.
CREATE DATABASE/CREATE DATABASE AT NODE/ADD NODE/DROP NODE VERIFY	After a certain point, these APIs are not interruptible. If the interrupt is received before this point, the database is not created. If the interrupt is received after this point, the interrupt is ignored.
DROP DATABASE/DROP DATABASE AT NODE	None. These APIs complete.
Directory Services	Directory left in consistent state. Utility function may or may not be performed.
SQL Data Definition statements	Database transactions set to state existing prior to the SQL statement.
Other SQL statements	Database transactions set to state existing prior to the SQL statement.

See Also

“sqleisig - Install Signal Handler” on page 143.

sqleisig - Install Signal Handler

Installs the default interrupt (usually Control-C and/or Control-Break) signal handler. When this default handler detects an interrupt signal, it resets the signal and calls “sqleintr - Interrupt” on page 140.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Install Signal Handler */
/* ... */
SQL_API_RC SQL_API_FN
  sqleisig (
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Install Signal Handler */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgisig (
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqleisig - Install Signal Handler

REXX API Syntax

```
INSTALL SIGNAL HANDLER
```

Sample Programs

C	\sqllib\samples\c\dbcmt.c
COBOL	\sqllib\samples\cobol\ish.cbl
FORTRAN	\sqllib\samples\fortran\ish.f
REXX	\sqllib\samples\rexx\dbcmt.cmd

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 "sqleintr - Interrupt" on page 140 can be developed. Use either the operating system call or the language-specific library signal function. "sqleintr - Interrupt" on page 140 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.

See Also

"sqleintr - Interrupt" on page 140.

sqlemgdb - Migrate Database

Converts previous versions of DB2 databases to current formats. Following are the database releases that are supported in the DB2 V5.0 database migration process:

- DB2 for OS/2 Version 1.x and Version 2.x to Version 5.0
- DB2 for AIX Version 1.x and Version 2.x to Version 5.0
- DB2 for HP-UX Version 2.x to Version 5.0
- DB2 for Solaris Version 2.x to Version 5.0
- DB2 for Windows NT Version 2.x to Version 5.0
- DB2 Parallel Edition Version 1.x to Version 5.0.

Authorization

sysadm

Required Connection

This API establishes a database connection.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Migrate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlemgdb (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pPassword,
    struct sqlca * pSqlca);
/* ... */
```

sqlemgdb - Migrate Database

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Migrate Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmgdb (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pPassword,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API Parameters

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

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

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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

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

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

REXX API Syntax

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API Parameters

dbalias
: Alias of the database to be migrated.

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

password
: Password used to authenticate the user name.

Sample Programs

C	\sqllib\samples\c\migrate.c
COBOL	\sqllib\samples\cobol\migrate.cbl
FORTRAN	\sqllib\samples\fortran\migrate.f
REXX	\sqllib\samples\rexx\migrate.cmd

Usage Notes

This API will only migrate a database to a newer version, and cannot be used to convert a migrated database to its previous version.

The database must be cataloged before migration.

For detailed information about database migration, see one of the *Quick Beginnings* books.

sqlencls - Close Node Directory Scan

Frees the resources that are allocated by “sqlenops - Open Node Directory Scan” on page 153.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Close Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlencls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Close Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgncls (
    unsigned short Handle,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Handle
: Input. Identifier returned from the associated OPEN NODE DIRECTORY SCAN API.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

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.

Sample Programs

C	\sqllib\samples\c\nodecat.sqc
COBOL	\sqllib\samples\cobol\nodecat.sqb
FORTRAN	\sqllib\samples\fortran\nodecat.sqf
REXX	\sqllib\samples\rexx\nodecat.cmd

See Also

"sqlengne - Get Next Node Directory Entry" on page 150
"sqlenops - Open Node Directory Scan" on page 153.

sqlengne - Get Next Node Directory Entry

Returns the next entry in the node directory after “sqlenops - Open Node Directory Scan” on page 153 is called. Subsequent calls to this API return additional entries.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Get Next Node Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlengne (
    unsigned short Handle,
    struct sqleninfo ** ppNodeDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Get Next Node Directory Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgngne (
    unsigned short Handle,
    struct sqleninfo ** ppNodeDirEntry,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Handle
: Input. Identifier returned from “sqlenops - Open Node Directory Scan” on page 153.

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" on page 153. For more information about the *sqleninfo* structure, see "SQLENINFO" on page 397.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

```
GET NODE DIRECTORY ENTRY :scanid [USING :value]
```

REXX API Parameters

scanid
: A REXX host variable containing the identifier returned from the OPEN NODE DIRECTORY SCAN API.

value
: A compound REXX host variable to which the node entry information is returned. If no name is given, the name `SQLNINFO` is used. In the following, XXX represents the host variable name (the corresponding field names are taken from the structure returned by the API):

XXX.0	Number of elements in the variable (always 16)
XXX.1	NODENAME
XXX.2	LOCALLU
XXX.3	PARTNERLU
XXX.4	MODE
XXX.5	COMMENT
XXX.6	RESERVED
XXX.7	PROTOCOL (protocol type)
XXX.8	ADAPTER (NetBIOS adapter #)
XXX.9	RESERVED
XXX.10	SYMDESTNAME (symbolic destination name)
XXX.11	SECURITY (security type)
XXX.12	HOSTNAME
XXX.13	SERVICENAME

XXX.14	FILESERVER
XXX.15	OBJECTNAME
XXX.16	INSTANCE (local instance name).

Sample Programs

C	\sqllib\samples\c\nodecat.c
COBOL	\sqllib\samples\cobol\nodecat.cbl
FORTRAN	\sqllib\samples\fortran\nodecat.f
REXX	\sqllib\samples\rexx\nodecat.cmd

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 (*pNumEntries* is returned by "sqlenops - Open Node Directory Scan" on page 153).

See Also

"sqlencls - Close Node Directory Scan" on page 148
"sqlenops - Open Node Directory Scan" on page 153.

sqlenops - Open Node Directory Scan

Stores a copy in memory of the node directory, and returns the number of entries. This is a snapshot of the directory at the time the directory is opened. This copy is not updated, even if the directory itself is changed later.

Use “sqlengne - Get Next Node Directory Entry” on page 150 to advance through the node directory and examine information about the node entries. Close the scan using “sqlencls - Close Node Directory Scan” on page 148. This removes the copy of the directory from memory.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Open Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlenops (
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Open Node Directory Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgnops (
    unsigned short * pHandle,
    unsigned short * pNumEntries,
    struct sqlca * pSqlca);
/* ... */
```

sqlenops - Open Node Directory Scan

API Parameters

pHandle
: Output. Identifier returned from this API. This identifier must be passed to “sqlengne - Get Next Node Directory Entry” on page 150, and “sqlencls - Close Node Directory Scan” on page 148.

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. For more information about this structure, see “SQLCA” on page 355.

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.

Sample Programs

C	\sqllib\samples\c\nodecat.c
COBOL	\sqllib\samples\cobol\nodecat.cbl
FORTRAN	\sqllib\samples\fortran\nodecat.f
REXX	\sqllib\samples\rexx\nodecat.cmd

Usage Notes

Storage allocated by this API is freed by calling “sqlencls - Close Node Directory Scan” on page 148.

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.

See Also

“sqlencls - Close Node Directory Scan” on page 148
“sqlengne - Get Next Node Directory Entry” on page 150.

sqlepstart - Start Database Manager

Starts the current database manager instance background processes on a single node or on all the nodes defined in a multi-node environment.

This API is not valid on a client.

Scope

In a multi-node environment, this API affects all nodes that are listed in the `$HOME/sqllib/db2nodes.cfg` file, unless the *nodenum* parameter is used (see "SQLE-START-OPTIONS" on page 381).

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Start Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlepstart (
    struct sqle_start_options * pStartOptions,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Start Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgpstart (
    struct sqle_start_options * pStartOptions,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pStartOptions
A pointer to the *sqle_start_options* structure. This structure contains the start-up options. The pointer can be null. For more information about this structure, see “SQLE-START-OPTIONS” on page 381.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\dbstart.c
COBOL	\sqllib\samples\cobol\dbstart.cbl
FORTRAN	\sqllib\samples\fortran\dbstart.f
REXX	\sqllib\samples\rexx\dbstart.cmd

Usage Notes

It is not necessary to call this API on a client node. It is provided for compatibility with older clients, but it has no effect on the database manager.

Once started, the database manager instance runs until the user stops it, even if all application programs that were using it have ended.

If no parameters are specified in a multi-node database environment, the database manager is started on all parallel nodes specified in the node configuration file.

If the API call is still processing, ensure that the applicable nodes have started *before* issuing a request to the database.

The `db2cshrc` file is not supported and cannot be used to define the environment.

On UNIX platforms, **sqlepstart** supports the SIGINT and SIGALRM signals. The SIGINT signal is issued if CTRL+C is pressed. The SIGALRM signal is issued if the value specified for the *start_stop_time* database manager configuration parameter is reached. If either signal occurs, all in-progress startups are interrupted and a message (SQL1044N for SIGINT and SQL6037N for SIGALRM) is returned from each interrupted node to the `$HOME/sqllib/log/db2start.` *timestamp*`.log` error log file. Nodes that are already started are not affected. If CTRL+C is pressed on a node that is starting, **db2stop** must be issued on that node before an attempt is made to start it again.

See Also

"sqleaddn - Add Node" on page 65
"sqlepstp - Stop Database Manager" on page 159.

sqlepstp - Stop Database Manager

Stops the current database manager instance. Unless explicitly stopped, the database manager continues to be active. This API does not stop the database manager instance if any applications are connected to databases. If there are no database connections, but there are instance attachments, it forces the instance attachments and stops the database manager. This API also deactivates any outstanding database activations before stopping the database manager.

This API can also be used to drop a node from the `db2nodes.cfg` file (MPP systems only).

This API is not valid on a client.

Scope

In a multi-node environment, this API affects all nodes that are listed in the `$HOME/sqllib/db2nodes.cfg` file, unless the *nodenum* parameter is used (see “SQLEDBSTOPOPT” on page 392).

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Stop Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlepstp (
    struct sqledbstopopt * pStopOptions,
    struct sqlca * pSqlca);
/* ... */
```

sqlepstp - Stop Database Manager

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Stop Database Manager */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgpstp (
    struct sqledbstopopt * pStopOptions,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pStopOptions
: A pointer to the *sqledbstopopt* structure. This structure contains the stop options. The pointer can be null. For more information about this structure, see “SQLEDBSTOPOPT” on page 392.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\dbstop.c
COBOL	\sqllib\samples\cobol\dbstop.cbl
FORTRAN	\sqllib\samples\fortran\dbstop.f
REXX	\sqllib\samples\rexx\dbstop.cmd

Usage Notes

It is not necessary to call this API on a client node. It is provided for compatibility with older clients, but it has no effect on the database manager.

Once started, the database manager instance runs until the user stops it, even if all application programs that were using it have ended.

If the database manager cannot be stopped because application programs are still connected to databases, use “sqlefrce - Force Application” on page 119 to disconnect all users first, or call the **sqlepstp** API again with the FORCE option.

The following information currently applies to multiple node environments only:

- If no parameters are specified, the database manager is stopped on each node listed in the node configuration file. The `db2diag.log` file may contain messages to indicate that other nodes are shutting down.
- Any nodes added to the MPP system since the previous call to **sqlepstp** will be updated in the `db2nodes.cfg` file.
- On UNIX platforms, this API supports the SIGALRM signal, which is issued if the value specified for the *start_stop_time* database manager configuration parameter is reached. If this signal occurs, all in-progress stops are interrupted, and message SQL6037N is returned from each interrupted node to the `$HOME/sqllib/log/db2stop.` *timestamp*`.log` error log file. Nodes that are already stopped are not affected.
- The `db2cshrc` file is not supported and cannot be specified as the value for the PROFILE parameter.

See Also

"sqle_deactivate_db - Deactivate Database" on page 62
"sqledrpn - Drop Node Verify" on page 113
"sqlefrce - Force Application" on page 119
"sqlepstart - Start Database Manager" on page 156.

sqleqryc - Query Client

Returns current connection settings for an application process. For information about the applicable connection settings and their values, see “SQLE-CONN-SETTING” on page 367.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Query Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqleqryc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Query Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgqryc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pConnectionSettings
: Input/Output. A pointer to an *sqle_conn_setting* structure, which specifies connection setting types and values. The user defines an array of *NumSettings* connection settings structures, and sets the *type* field of each element in this array to indicate one of the five possible connection settings options. Upon return, the *value* field of each element contains the current setting of the option specified. For more information about this structure, see “SQLE-CONN-SETTING” on page 367.

NumSettings
: Input. Any integer (from 0 to 5) representing the number of connection option values to be returned.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
QUERY CLIENT INTO :output
```

REXX API Parameters

output
: A compound REXX host variable containing information about the current connection settings of the application process. In the following, XXX represents the host variable name.

XXX.1	Current connection setting for the CONNECTION type
XXX.2	Current connection setting for the SQLRULES
XXX.3	Current connection setting indicating which connections will be released when a COMMIT is issued.
XXX.4	Current connection setting of the SYNCPOINT option. Indicates whether a transaction manager should be used to enforce two-phase commit semantics, whether the database manager should ensure that there is only one database being updated when multiple databases are accessed within a single transaction, or whether neither of these options is to be used.
XXX.5	Current connection setting for the maximum number of concurrent connections for a NETBIOS adapter.
XXX.6	Current connection setting for deferred PREPARE.

sqleqryc - Query Client

Sample Programs

C	\sqllib\samples\c\client.c
COBOL	\sqllib\samples\cobol\client.cbl
FORTRAN	\sqllib\samples\fortran\client.f
REXX	\sqllib\samples\rexx\client.cmd

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.

For information about distributed unit of work (DUOW), see the *Administration Guide*.

See Also

"sqlesetc - Set Client" on page 176.

sqleregs - Register

Registers the DB2 server on the network server. The DB2 server's network address is stored in a specified registry on the file server, where it can be retrieved by a client application that uses the IPX/SPX communication protocol.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Register */
/* ... */
SQL_API_RC SQL_API_FN
  sqleregs (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Register */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgregs (
    unsigned short Registry,
    void * pRegisterInfo,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Registry
: Input. Indicates where on the network file server to register the DB2 server. In this release, the only supported value is `SQL_NWBINDERY` (NetWare file server bindery, defined in `sqlenv`).

pRegisterInfo
: Input. A pointer to the *sqle_reg_nwbindery* structure. In the structure, the caller specifies a user name and password that are valid on the network file server. For more information about this structure, see "SQLE-REG-NWBINDERY" on page 380.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\regder.c
COBOL	\sqllib\samples\cobol\regder.cbl
FORTRAN	\sqllib\samples\fortran\regder.f

Usage Notes

This API determines the IPX/SPX address of the DB2 server machine (the machine from which it was called), and then creates an object in the NetWare file server bindery using the value for *objectname* specified in the database manager configuration file. The IPX/SPX address of the DB2 server is stored as a property in that object. In order for a client to connect or attach to a DB2 database using IPX/SPX file server addressing, it must catalog an IPX/SPX node (using the same FILESERVER and OBJECTNAME specified on the server) in the node directory.

The specified NetWare user name and password must have supervisory or equivalent authority.

This API *must* be issued locally from a DB2 server. It is not supported remotely.

After installation and configuration of DB2, the DB2 server should be registered once on the network file server (unless only *direct addressing* will be used by IPX/SPX clients to connect to this DB2 server). After that, if the IPX/SPX fields are reconfigured, or the DB2 server's IPX/SPX internetwork address changes, deregister the DB2 server on the network file server before making the changes, and then register it again after the changes have been made.

See Also

"sqledreg - Deregister" on page 108.

sqlerstd - Restart Database

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

Scope

This API affects only the node on which it is executed.

Authorization

None

Required Connection

This API establishes a database connection.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Restart Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlerstd (
    _SQLOLDCHAR * pDbAlias,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pPassword,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Restart Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrstd (
    unsigned short PasswordLen,
    unsigned short UserNameLen,
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pPassword,
    _SQLOLDCHAR * pUserName,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API Parameters

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

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

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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

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

pDbAlias
: Input. A string containing the alias of the database that is to be restarted.

REXX API Syntax

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

sqlerstd - Restart Database

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.

Sample Programs

C	\sqllib\samples\c\restart.c
COBOL	\sqllib\samples\cobol\restart.cbl
FORTRAN	\sqllib\samples\fortran\restart.f
REXX	\sqllib\samples\rexx\restart.cmd

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 RESTART DATABASE must be issued before the database can be used again. Use the transaction APIs (see Appendix B, “Transaction APIs” on page 447) to generate a list of indoubt transactions. For more information about indoubt transactions, see the *Administration Guide*.

If the database is only restarted on a single node within an MPP system, a message may be returned on a subsequent database query indicating that the database needs to be restarted. This occurs because the database on a node on which the query depends must also be restarted. Restarting the database on all nodes solves the problem.

See Also

CONNECT TO statement in the *SQL Reference*.

sqlesact - Set Accounting String

Provides accounting information that will be sent to a DRDA server with the application's next connect request.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Set Accounting String */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesact (
    char * pAccountingString,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Set Accounting String */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsact (
    unsigned short AccountingStringLen,
    char * pAccountingString,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

AccountingStringLen
: Input. A 2-byte unsigned integer representing the length in bytes of the accounting string.

pAccountingString
: Input. A string containing the accounting data.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

Sample Programs

C	\sqllib\samples\c\setact.c
COBOL	\sqllib\samples\cobol\setact.cbl
FORTRAN	\sqllib\samples\fortran\setact.f

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 (_).

See Also

The *DB2 Connect User's Guide* contains more information about the accounting string and the DRDA servers that support it.

sqlesdeg - Set Runtime Degree

Sets the maximum run time degree of intra-partition parallelism for SQL statements for specified active applications. It has no effect on CREATE INDEX parallelism.

Scope

This API affects all nodes that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

Instance. To change the maximum run time degree of parallelism on a remote server, it is first necessary to attach to that server. If no attachment exists, the SET RUNTIME DEGREE statement fails.

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Set Runtime Degree */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesdeg (
    long NumAgentIds,
    unsigned long * pAgentIds,
    long Degree,
    struct sqlca * pSqlca);
/* ... */
```

sqlesdeg - Set Runtime Degree

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Set Runtime Degree */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsdeg (
    struct sqlca * pSqlca,
    long Degree,
    unsigned long * pAgentIds,
    long NumAgentIds);
/* ... */
```

API Parameters

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Degree
: Input. The new value for the maximum run time degree of parallelism. The value must be in the range 1 to 32767.

pAgentIds
: Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding application. To list the agent IDs of the active applications, use “sqlmonss - Get Snapshot” on page 215.

NumAgentIds
: Input. An integer representing the total number of active applications to which the new degree value will apply. This number should be the same as the number of elements in the array of agent IDs.

  If this parameter is set to `SQL_ALL_USERS` (defined in `sqlenv`), the new degree will apply to all active applications. If it is set to zero, an error is returned.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C \sqllib\samples\c\setrundg.c

Usage Notes

The database system monitor functions are used to gather the agent IDs and degrees of active applications. For more information, see the *System Monitor Guide and Reference*.

Minimal validation is performed on the array of agent IDs. The user must ensure that the pointer points to an array containing the total number of elements specified. If *NumAgentIds* is set to `SQL_ALL_USERS`, the array is ignored.

If one or more specified agent IDs cannot be found, the unknown agent IDs are ignored, and the function continues. No error is returned. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and the API is called.

Agent IDs are recycled, and are used to change the degree of parallelism for applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the new degree of parallelism will be modified for the wrong user.

See Also

"sqlmonss - Get Snapshot" on page 215.

sqlesetc - Set Client

Specifies connection settings for the application. For information about the applicable connection settings and their values, see "SQLE-CONN-SETTING" on page 367.

Authorization

None

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Set Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqlesetc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Set Client */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgsetc (
    struct sqle_conn_setting * pConnectionSettings,
    unsigned short NumSettings,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pConnectionSettings

Input. A pointer to the *sqle_conn_setting* structure, which specifies connection setting types and values. Allocate an array of *NumSettings* *sqle_conn_setting* structures. Set the *type* field of each element in this array to indicate the connection option to set. Set the *value* field to the desired value for the option. For more information about this structure, see "SQLE-CONN-SETTING" on page 367.

NumSettings
Input. Any integer (from 0 to 5) representing the number of connection options to set.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

```
SET CLIENT USING :values
```

REXX API Parameters

values
A compound REXX host variable containing the connection settings for the application process. In the following, XXX represents the host variable name.

XXX.0 Number of connection settings to be established

XXX.1 Specifies how to set up the CONNECTION type. The valid values are:

1 Type 1 CONNECT

2 Type 2 CONNECT

XXX.2 Specifies how to set up the SQLRULES. The valid values are:

DB2 Process type 2 CONNECT according to the DB2 rules

STD Process type 2 CONNECT according to the Standard rules

XXX.3 Specifies how to set up the scope of disconnection to databases at commit. The valid values are:

EXPLICIT	Disconnect only those marked by the SQL RELEASE statement
CONDITIONAL	Disconnect only those that have no open WITH HOLD cursors
AUTOMATIC	Disconnect all connections

XXX.4 Specifies how to set up the coordination among multiple database connections during commits or rollbacks. The valid values are:

TWOPHASE	Use Transaction Manager (TM) to coordinate two-phase commits

ONEPHASE	Use one-phase commit
NONE	Do not enforce single updater and multiple reader

XXX.5 Specifies the maximum number of concurrent connections for a NETBIOS adapter.

XXX.6 Specifies when to execute the PREPARE statement. The valid values are:

NO	The PREPARE statement will be executed at the time it is issued
YES	The PREPARE statement will not be executed until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued. However, the PREPARE INTO statement is not deferred
ALL	Same as YES, except that the PREPARE INTO statement is also deferred

Sample Programs

C	\sqllib\samples\c\client.c
COBOL	\sqllib\samples\cobol\client.cbl
FORTRAN	\sqllib\samples\fortran\client.f
REXX	\sqllib\samples\rexx\client.cmd

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.

For information about distributed unit of work (DUOW), see the *Administration Guide*.

See Also

“sqleqryc - Query Client” on page 162.

sqleuncd - Uncatalog Database

Deletes an entry from the system database directory.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Uncatalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncd (
    _SQLOLDCHAR * pDbAlias,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Uncatalog Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncd (
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pDbAlias);
/* ... */
```

API Parameters

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDbAlias
: Input. A string containing the database alias that is to be uncataloged.

REXX API Syntax

```
UNCATALOG DATABASE dbname
```

REXX API Parameters

dbname
: Alias of the database to be uncataloged.

Sample Programs

C	\sqllib\samples\c\dbcat.c
COBOL	\sqllib\samples\cobol\dbcat.cbl
FORTRAN	\sqllib\samples\fortran\dbcat.f
REXX	\sqllib\samples\rexx\dbcat.cmd

Usage Notes

Only entries in the system database directory can be uncataloged. Entries in the local database directory can be deleted using “sqledrpd - Drop Database” on page 110.

To recatalog the database, use “sqlecadb - Catalog Database” on page 72.

To list the databases that are cataloged on a node, use “sqledosd - Open Database Directory Scan” on page 103, “sqledgne - Get Next Database Directory Entry” on page 100, and “sqledcls - Close Database Directory Scan” on page 98.

The authentication type of a database, used when communicating with a down-level server, can be changed by first uncataloging the database, and then cataloging it again with a different type.

If directory caching is enabled (see the configuration parameter *dir_cache* in “sqlfxsys - Get Database Manager Configuration” on page 204), database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

See Also

“sqlecadb - Catalog Database” on page 72
“sqledcls - Close Database Directory Scan” on page 98
“sqledgne - Get Next Database Directory Entry” on page 100
“sqledosd - Open Database Directory Scan” on page 103.

sqleuncn - Uncatalog Node

Deletes an entry from the node directory.

Authorization

One of the following:

sysadm
sysctrl

Required Connection

None

API Include File

sqlenv.h

C API Syntax

```
/* File: sqlenv.h */
/* API: Uncatalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqleuncn (
    _SQLOLDCHAR * pNodeName,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlenv.h */
/* API: Uncatalog Node */
/* ... */
SQL_API_RC SQL_API_FN
  sqlguncn (
    unsigned short NodeNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pNodeName);
/* ... */
```

API Parameters

NodeNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the node name.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

pNodeName
: Input. A string containing the name of the node to be uncataloged.

REXX API Syntax

```
UNCATALOG NODE nodename
```

REXX API Parameters

nodename
: Name of the node to be uncataloged.

Sample Programs

C	\sqllib\samples\c\nodecat.c
COBOL	\sqllib\samples\cobol\nodecat.cbl
FORTRAN	\sqllib\samples\fortran\nodecat.f
REXX	\sqllib\samples\rexx\nodecat.cmd

Usage Notes

To recatalog the node, use "sqlectnd - Catalog Node" on page 89.

To list the nodes that are cataloged, use "sqlenops - Open Node Directory Scan" on page 153, "sqlengne - Get Next Node Directory Entry" on page 150, and "sqlencls - Close Node Directory Scan" on page 148.

If directory caching is enabled (see the configuration parameter *dir_cache* in "sqlfxsys - Get Database Manager Configuration" on page 204), database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

See Also

"sqlectnd - Catalog Node" on page 89
"sqlencls - Close Node Directory Scan" on page 148
"sqlengne - Get Next Node Directory Entry" on page 150
"sqlenops - Open Node Directory Scan" on page 153.

sqlfddb - Get Database Configuration Defaults

Returns the default values of individual entries in a database configuration file.

Authorization

None

Required Connection

Instance. It is not necessary to call ATTACH before getting the configuration of 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

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfddb (
    char * pDbAlias,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgddb (
    unsigned short DbAliasLen,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API Parameters

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

NumItems
: Input. Number of entries to be returned. The minimum valid value is 1.

pItemList
: Input/Output. Pointer to an array of *NumItems sqlfupd* structures, each containing a token field indicating which value to return, and a pointer field indicating where to place the configuration value. For more information about this structure, see "SQLFUPD" on page 400.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

pDbAlias
: Input. A string containing the database alias.

Sample Programs

C	\sqllib\samples\c\d_dbconf.c
COBOL	\sqllib\samples\cobol\d_dbconf.cbl
FORTRAN	\sqllib\samples\fortran\d_dbconf.f

Usage Notes

The application is responsible for allocating sufficient memory for each data element returned. For example, the value returned for *newlogpath* can be up to 242 bytes in length.

DB2 returns the current value of non-updateable parameters.

If an error occurs, the information returned is not valid. If the configuration file is invalid, an error message is returned. The database must be restored from a backup version.

To set the database configuration parameters to the recommended database manager defaults, use "sqlfrdb - Reset Database Configuration" on page 188.

For a brief description of the database configuration parameters, see the *Command Reference*. For more information about tuning these parameters, see the *Administration Guide*.

See Also

"sqlfrdb - Reset Database Configuration" on page 188
"sqlfudb - Update Database Configuration" on page 194
"sqlfxdb - Get Database Configuration" on page 201.

sqlfdsys - Get Database Manager Configuration Defaults

Returns the default values of individual entries in the database manager configuration file.

Authorization

None

Required Connection

None or instance. An instance attachment is not required to perform database manager configuration operations at the current instance (as defined by the value of the **DB2INSTANCE** environment variable), but is required to perform database manager configuration operations at other instances. To display the database manager configuration for another instance, it is necessary to first attach to that instance.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfdsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration Defaults */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

NumItems
: Input. Number of entries being returned. The minimum valid value is 1.

pItemList
: Input/Output. Pointer to an array of *NumItems sqlfupd* structures, each containing a token field indicating which value to return, and a pointer field indicating where to place the configuration value. For more information about this structure, see “SQLFUPD” on page 400.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Sample Programs

C	\sqllib\samples\c\d_dbmcon.c
COBOL	\sqllib\samples\cobol\d_dbmcon.cbl
FORTRAN	\sqllib\samples\fortran\d_dbmcon.f

Usage Notes

If an attachment to a remote instance (or a different local instance) exists, the default database manager configuration parameters for the attached server are returned; otherwise, the local default database manager configuration parameters are returned.

If an error occurs, the information returned is not valid. If the configuration file is invalid, an error message is returned. The user must again install the database manager to recover.

The current value of non-updateable parameters is returned as the default.

To set the database manager configuration parameters to the recommended database manager defaults, use “sqlfrsys - Reset Database Manager Configuration” on page 191.

For a brief description of the database manager configuration parameters, see the *Command Reference*. For more information about tuning these parameters, see the *Administration Guide*.

See Also

“sqlfrsys - Reset Database Manager Configuration” on page 191
“sqlfusys - Update Database Manager Configuration” on page 198
“sqlfxsys - Get Database Manager Configuration” on page 204.

sqlfrdb - Reset Database Configuration

Resets the configuration file of a specific database to the system defaults.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. An explicit attachment is not required. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the call.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Reset Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfrdb (
    _SQLOLDCHAR * pDbAlias,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Reset Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrdb (
    unsigned short DbAliasLen,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API Parameters

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

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDbAlias
: Input. A string containing the database alias.

REXX API Syntax

```
RESET DATABASE CONFIGURATION FOR dbname
```

REXX API Parameters

dbname
: Alias of the database associated with the configuration file.

Sample Programs

C	\sqllib\samples\c\dbconf.c
COBOL	\sqllib\samples\cobol\dbconf.cbl
FORTRAN	\sqllib\samples\fortran\dbconf.f
REXX	\sqllib\samples\rexx\dbconf.cmd

Usage Notes

This API resets the entire configuration (except for non-updateable parameters).

To view or print a list of the current database configuration parameters for a database, use “sqlfxdb - Get Database Configuration” on page 201.

To view the default values for database configuration parameters, use “sqlfddb - Get Database Configuration Defaults” on page 184.

To change the value of a configurable parameter, use “sqlfudb - Update Database Configuration” on page 194.

Changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur.

If an error occurs, the database configuration file does not change.

The database configuration file cannot be reset if the checksum is invalid. This may occur if the database configuration file is changed without using the appropriate API. If this happens, the database must be restored to reset the database configuration file.

sqlfrdb - Reset Database Configuration

For a brief description of the database configuration parameters, see the *Command Reference*. For more information about these parameters, see the *Administration Guide*.

See Also

"sqlfddb - Get Database Configuration Defaults" on page 184
"sqlfudb - Update Database Configuration" on page 194
"sqlfxdb - Get Database Configuration" on page 201.

sqlfrsys - Reset Database Manager Configuration

Resets the parameters in the database manager configuration file to the system defaults.

Authorization

sysadm

Required Connection

None or instance. An instance attachment is not required to perform database manager configuration operations at the current instance (as defined by the value of the **DB2INSTANCE** environment variable), but is required to perform database manager configuration operations at other instances. To reset the database manager configuration for another instance, it is necessary to first attach to that instance.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Reset Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfrsys (
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Reset Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrsys (
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqlfrsys - Reset Database Manager Configuration

REXX API Syntax

```
RESET DATABASE MANAGER CONFIGURATION
```

Sample Programs

C	\sqllib\samples\c\dbmconf.c
COBOL	\sqllib\samples\cobol\dbmconf.cbl
FORTRAN	\sqllib\samples\fortran\dbmconf.f
REXX	\sqllib\samples\rexx\dbmconf.cmd

Usage Notes

If an attachment to a remote instance (or a different local instance) exists, the database manager configuration parameters for the attached server are reset; otherwise, the local database manager configuration parameters are reset.

This API resets the entire configuration (except for non-updateable parameters).

To view or print a list of the current database manager configuration parameters, use "sqlfxsys - Get Database Manager Configuration" on page 204.

To view the default values for database manager configuration parameters, use "sqlfdsys - Get Database Manager Configuration Defaults" on page 186.

To change the value of a configurable parameter, use "sqlfusys - Update Database Manager Configuration" on page 198.

Changes to the database manager configuration file become effective only after they are loaded into memory. For a server configuration parameter, this occurs during execution of **db2start**. For a client configuration parameter, this occurs when the application is restarted.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be reset if the checksum is invalid. This may occur if the database manager configuration file is changed without using the appropriate API. If this happens, the database manager must be installed again to reset the database manager configuration file.

For a brief description of the database manager configuration parameters, see the *Command Reference*. For more information about these parameters, see the *Administration Guide*.

See Also

"sqlfdsys - Get Database Manager Configuration Defaults" on page 186
"sqlfusys - Update Database Manager Configuration" on page 198
"sqlfxsys - Get Database Manager Configuration" on page 204.

sqlfudb - Update Database Configuration

Modifies individual entries in a specific database configuration file.

A database configuration file resides on every node on which the database has been created.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. An explicit attachment is not required. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the call.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Update Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfudb (
    _SQLOLDCHAR * pDbAlias,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Update Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgudb (
    unsigned short DbAliasLen,
    unsigned short NumItems,
    unsigned short * pItemListLens,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API Parameters

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

NumItems
: Input. Number of entries being modified. The minimum valid value is 1.

pItemListLens
: Input. An array of 2-byte unsigned integers representing the length of each of the new configuration field values in the *pItemList*. It is necessary to provide lengths for those fields that contain strings only, such as *newlogpath*. If, for example, *newlogpath* is the fifth element in the *pItemList* array, its length must be the fifth element in the *pItemListLens* array.

pItemList
: Input. Pointer to an array of *NumItems* *sqlfupd* structures, each containing a token field indicating which value to update, and a pointer field indicating the new value. For more information about this structure, see "SQLFUPD" on page 400.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

pDbAlias
: Input. A string containing the database alias.

REXX API Syntax

```
UPDATE DATABASE CONFIGURATION FOR dbname USING :values
```

REXX API Parameters

dbname
: Alias of the database associated with the configuration file.

values
: A compound REXX host variable containing tokens indicating which configuration fields are to be modified. The application provides the token and the new value for each field. The following are elements of a variable, where XXX represents the host variable name:

XXX.0	Twice the number of fields supplied (number of data elements in the remainder of the variable)
XXX.1	First token
XXX.2	Value supplied for the first field
XXX.3	Second token
XXX.4	Value supplied for the second field
XXX.5	and so on.

Sample Programs

C	\sqllib\samples\c\dbconf.c
COBOL	\sqllib\samples\cobol\dbconf.cbl
FORTRAN	\sqllib\samples\fortran\dbconf.f
REXX	\sqllib\samples\rexx\dbconf.cmd

Usage Notes

To view or print a list of the database configuration parameters, use “sqlfxdb - Get Database Configuration” on page 201.

To view the default values for database configuration parameters, use “sqlfddb - Get Database Configuration Defaults” on page 184.

To reset the database configuration parameters to the recommended defaults, use “sqlfrdb - Reset Database Configuration” on page 188.

The default values of these parameters may differ for each type of database node configured (server, client, or server with remote clients). See the *Administration Guide* for the ranges and the default values that can be set on each node type. The valid *token* values for each configuration entry are listed in Table 42 on page 400.

Not all parameters can be updated.

Changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur.

If an error occurs, the database configuration file does not change.

The database configuration file cannot be updated if the checksum is invalid. This may occur if the database configuration file is changed without using the appropriate API. If this happens, the database must be restored to reset the database configuration file.

For a brief description of the database configuration parameters, see the *Command Reference*. For more information about these parameters, see the *Administration Guide*.

See Also

"sqlfddb - Get Database Configuration Defaults" on page 184
"sqlfrdb - Reset Database Configuration" on page 188
"sqlfxdb - Get Database Configuration" on page 201.

sqlfusys - Update Database Manager Configuration

Modifies individual entries in the database manager configuration file.

Authorization

sysadm

Required Connection

None or instance. An instance attachment is not required to perform database manager configuration operations at the current instance (as defined by the value of the **DB2INSTANCE** environment variable), but is required to perform database manager configuration operations at other instances. To update the database manager configuration for another instance, it is necessary to first attach to that instance.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Update Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfusys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Update Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgusys (
    unsigned short NumItems,
    unsigned short * pItemListLens,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

NumItems
: Input. Number of entries being modified. The minimum valid value is 1.

pItemListLens
: Input. An array of 2-byte unsigned integers representing the length of each of the new configuration field values in the *pItemList*. It is necessary to provide lengths for those fields that contain strings only, such as *dftdbpath*. If, for example, *dftdbpath* is the fifth element in the *pItemList* array, its length must be the fifth element in the *pItemListLens* array.

pItemList
: Input. Pointer to an array of *NumItems sqlfupd* structures, each containing a token field indicating which value to update, and a pointer field indicating the new value. For more information about this structure, see “SQLFUPD” on page 400.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
UPDATE DATABASE MANAGER CONFIGURATION USING :values
```

REXX API Parameters

values
: A compound REXX host variable containing tokens that indicate the configuration fields to be modified. The application provides the token and the new value for each field. The following are elements of a variable, where XXX represents the host variable name:

XXX.0	Number of elements in the variable. This value is two times the number of fields to modify.
XXX.1	First token
XXX.2	New value for the first field
XXX.3	Second token
XXX.4	New value for the second field
XXX.5	and so on.

Sample Programs

C	\sqllib\samples\c\dbmconf.c
COBOL	\sqllib\samples\cobol\dbmconf.cbl
FORTRAN	\sqllib\samples\fortran\dbmconf.f
REXX	\sqllib\samples\rexx\dbmconf.cmd

Usage Notes

If an attachment to a remote instance (or a different local instance) exists, the database manager configuration parameters for the attached server are updated; otherwise, the local database manager configuration parameters are updated.

To view or print a list of the database manager configuration parameters, use "sqlfxsys - Get Database Manager Configuration" on page 204.

To reset the database manager configuration parameters to the recommended database manager defaults, use "sqlfrsys - Reset Database Manager Configuration" on page 191.

The default values of these parameters may differ for each type of database node configured (server, client, or server with remote clients). See the *Administration Guide* for the ranges and the default values that can be set on each node type. The valid *token* values for each configuration entry are listed in Table 44 on page 403.

Not all parameters can be updated.

Changes to the database manager configuration file become effective only after they are loaded into memory. For a server configuration parameter, this occurs during execution of **db2start**. For a client configuration parameter, this occurs when the application is restarted.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be updated if the checksum is invalid. This may occur if the database manager configuration file is changed without using the appropriate API. If this happens, the database manager must be reinstalled to reset the database manager configuration file.

For a brief description of the database manager configuration parameters, see the *Command Reference*. For more information about these parameters, see the *Administration Guide*.

See Also

"sqlfdsys - Get Database Manager Configuration Defaults" on page 186
"sqlfrsys - Reset Database Manager Configuration" on page 191
"sqlfxsys - Get Database Manager Configuration" on page 204.

sqlfxdb - Get Database Configuration

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

For a brief description of the database configuration parameters, see the *Command Reference*. For detailed information about these parameters, see the *Administration Guide*.

Scope

This API returns information only for the node from which it is called.

Authorization

None

Required Connection

Instance. It is not necessary to call ATTACH before getting the configuration of 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

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfxdb (
    _SQLOLDCHAR * pDbAlias,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

sqlfxdb - Get Database Configuration

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgxdb (
    unsigned short DbAliasLen,
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca,
    char * pDbAlias);
/* ... */
```

API Parameters

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

NumItems
: Input. Number of entries to be returned. The minimum valid value is 1.

pItemList
: Input/Output. Pointer to an array of *NumItem sqlfupd* structures, each containing a token field indicating which value to return, and a pointer field indicating where to place the configuration value. For more information about this structure, see “SQLFUPD” on page 400.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pDbAlias
: Input. A string containing the database alias.

REXX API Syntax

```
GET DATABASE CONFIGURATION FOR database_alias USING :values
```

REXX API Parameters

database_alias
: Alias of the database associated with a specific database configuration file.

values
: A compound REXX host variable containing tokens that indicate the configuration fields to be returned. The application provides the token and the API returns the value. The following are elements of a variable, where XXX represents the host variable name:

XXX.0	Twice the number of fields returned (number of data elements in the remainder of the variable)
XXX.1	First token
XXX.2	Value returned for the first field
XXX.3	Second token
XXX.4	Value returned for the second field
XXX.5	and so on.

Sample Programs

C	\sqllib\samples\c\dbconf.c
COBOL	\sqllib\samples\cobol\dbconf.cbl
FORTRAN	\sqllib\samples\fortran\dbconf.f
REXX	\sqllib\samples\rexx\dbconf.cmd

Usage Notes

Entries in the database configuration file that are not listed in the token values for *pItemList* are not accessible to the application.

The application is responsible for allocating sufficient memory for each data element returned. For example, the value returned for *newlogpath* can be up to 242 bytes in length.

If an error occurs, the information returned is not valid. If the configuration file is invalid, an error message is returned. The database must be restored from a backup version.

To set the database configuration parameters to the database manager defaults, use "sqlfrdb - Reset Database Configuration" on page 188.

For more information about these parameters, see the *Administration Guide*.

See Also

"sqlfddb - Get Database Configuration Defaults" on page 184
"sqlfrdb - Reset Database Configuration" on page 188
"sqlfudb - Update Database Configuration" on page 194.

sqlfxsys - Get Database Manager Configuration

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

For a brief description of the database manager configuration parameters, see the *Command Reference*. For detailed information about these parameters, see the *Administration Guide*.

Authorization

None

Required Connection

An instance attachment is not required to perform database manager configuration operations at the current instance (as defined by the value of the **DB2INSTANCE** environment variable), but is required to perform database manager configuration operations at other instances. To display the database manager configuration for another instance, it is necessary to first attach to that instance.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlfxsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Database Manager Configuration */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgxsys (
    unsigned short NumItems,
    struct sqlfupd * pItemList,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

NumItems
: Input. Number of entries being modified. The minimum valid value is 1.

pItemList
: Input/Output. Pointer to an array of *NumItems sqlfupd* structures, each containing a token field indicating which value to return, and a pointer field indicating where to place the configuration value. For more information about this structure, see "SQLFUPD" on page 400.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

```
GET DATABASE MANAGER CONFIGURATION USING :values
```

REXX API Parameters

values
: A compound host variable containing tokens indicating the configuration fields to be returned. The application provides the token, and the API returns the value. XXX represents the host variable name:

XXX.0	The actual number of data elements in the remainder of the variable
XXX.1	First token
XXX.2	Value returned for the first field
XXX.3	Second token
XXX.4	Value returned for the second field
XXX.5	and so on.

Sample Programs

C	\sqllib\samples\c\dbmconf.c
COBOL	\sqllib\samples\cobol\dbmconf.cbl
FORTRAN	\sqllib\samples\fortran\dbmconf.f
REXX	\sqllib\samples\rexx\dbmconf.cmd

Usage Notes

If an attachment to a remote instance (or a different local instance) exists, the database manager configuration parameters for the attached server are returned; otherwise, the local database manager configuration parameters are returned.

The application is responsible for allocating sufficient memory for each data element returned. For example, the value returned for *dftdbpath* can be up to 215 bytes in length.

If an error occurs, the information returned is invalid. If the configuration file is invalid, an error message is returned. The user must install the database manager again to recover.

To set the configuration parameters to the default values shipped with the database manager, use “sqlfrsys - Reset Database Manager Configuration” on page 191.

For more information about these parameters, see the *Administration Guide*.

See Also

“sqlfdsys - Get Database Manager Configuration Defaults” on page 186
“sqlfrsys - Reset Database Manager Configuration” on page 191
“sqlfusys - Update Database Manager Configuration” on page 198.

sqlgaddr - Get Address

Places the address of a variable into another variable. It is used in host languages, such as FORTRAN and COBOL, that do not provide pointer manipulation.

Authorization

None

Required Connection

None

API Include File

sqlutil.h

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Address */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgaddr (
    char * pVariable,
    char ** ppOutputAddress);
/* ... */
```

API Parameters

pVariable
: Input. Variable whose address is to be returned.

ppOutputAddress
: Output. A 4-byte area into which the variable address is returned.

Sample Programs

COBOL	\sqllib\samples\cobol\dbmconf.sqb
FORTRAN	\sqllib\samples\fortran\dbmconf.sqf

Usage Notes

This API is used in the COBOL and FORTRAN languages only.

See Also

“sqlgdref - Dereference Address” on page 208.

sqlgdref - Dereference Address

Copies data from a buffer that is defined by a pointer, into a variable that is directly accessible by the application. It is used in host languages, such as FORTRAN and COBOL, that do not provide pointer manipulation. This API can be used to obtain results from APIs, such as “sqlengne - Get Next Node Directory Entry” on page 150, that return a pointer to the desired data.

Authorization

None

Required Connection

None

API Include File

sqlutil.h

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Dereference Address */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdref (
    unsigned int NumBytes,
    char * pTargetBuffer,
    char ** ppSourceBuffer);
/* ... */
```

API Parameters

NumBytes
: Input. An integer representing the number of bytes to be transferred.

pTargetBuffer
: Output. Area into which the data are moved.

ppSourceBuffer
: Input. A pointer to the area containing the desired data.

Sample Programs

COBOL	\sqllib\samples\cobol\nodecat.sqb
FORTRAN	\sqllib\samples\fortran\nodecat.sqf

Usage Notes

This API is used in the COBOL and FORTRAN languages only.

See Also

“sqlgaddr - Get Address” on page 207.

sqlgmcpy - Copy Memory

Copies data from one memory area to another. It is used in host languages, such as FORTRAN and COBOL, that do not provide memory block copy functions.

Authorization

None

Required Connection

None

API Include File

sqlutil.h

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Copy Memory */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgmcpy (
    void * pTargetBuffer,
    const void * pSource,
    unsigned long NumBytes);
/* ... */
```

API Parameters

pTargetBuffer
: Output. Area into which to move the data.

pSource
: Input. Area from which to move the data.

NumBytes
: Input. A 4-byte unsigned integer representing the number of bytes to be transferred.

Sample Programs

COBOL	\sqllib\samples\cobol\tspace.sqb
FORTRAN	\sqllib\samples\fortran\tspace.sqf

Usage Notes

This API is used in the COBOL and FORTRAN languages only.

See Also

“sqlgaddr - Get Address” on page 207.

sqlmon - Get/Update Monitor Switches

Selectively turns on or off switches for groups of monitor data to be collected by the database manager. Returns the current state of these switches for the application issuing the call.

Scope

This API only returns information for the node on which it is executed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. To display the settings for a remote instance, or for a different local instance, it is necessary to first attach to that instance.

API Include File

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Get/Update Monitor Switches */
/* ... */
int SQL_API_FN
  sqlmon (
    unsigned long         version,
    _SQLOLDCHAR           *reserved,
    sqlm_recording_group  group_states[],
    struct sqlca          *sqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlmon.h */
/* API: Get/Update Monitor Switches */
/* ... */
int SQL_API_FN
  sqlgmon (
    unsigned long          reserved_lgth,
    struct   sqlca         *sqlca,
    sqlm_recording_group   group_states[],
    _SQLOLDCHAR            *reserved,
    unsigned long          version);
/* ... */
```

API Parameters

reserved_lgth
: Reserved for future use. Users should set this value to zero.

sqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

group_states
: Input/Output. Pointer to an array of size SQLM_NUM_GROUPS (6). If the array size is less than six, an error message is returned. The user determines which element of the array corresponds to which switch by indexing it to the following symbolic statements (defined in `sqlmon.h`):

- SQLM_UOW_SW
- SQLM_STATEMENT_SW
- SQLM_TABLE_SW
- SQLM_BUFFER_POOL_SW
- SQLM_LOCK_SW
- SQLM_SORT_SW.

The array contains the following elements:

- An *input_state* element set to one of the following (defined in `sqlmon.h`):

  SQLM_ON
  Turns information group on.

  SQLM_OFF
  Turns information group off.

  SQLM_HOLD
  Leaves information group in its current state.

- An *output_state* element, containing current state information about the information group being monitored, is returned. `SQLM_ON` and `SQLM_OFF` indicate the state.

- A *start_time* element, indicating the time that the monitored group was turned on, is returned. If monitoring of this group is turned off, the time stamp is zero.

For more information about the *sqlm_recording_group* structure, see "SQLM-RECORDING-GROUP" on page 410, or the *System Monitor Guide and Reference*.

reserved
Reserved for future use. Users should set this value to NULL.

version
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`

If requesting data for a version higher than the current server, the database monitor only returns data for its level (see the *server_version* field in "SQLM-COLLECTED" on page 407).

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

Usage Notes

To obtain the status of the switches at the database manager level, call "sqlmonss - Get Snapshot" on page 215, specifying `SQMA_DB2` for *OBJ_TYPE* (get snapshot for database manager).

For detailed information about the use of the database monitor APIs, and for a summary of all database monitor data elements and monitoring groups, see the *System Monitor Guide and Reference*.

See Also

"sqlmonss - Get Snapshot" on page 215
"sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer" on page 218
"sqlmrset - Reset Monitor" on page 221.

sqlmonss - Get Snapshot

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

Scope

This API returns information only for the node on which it is issued.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. 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

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Get Snapshot */
/* ... */
int SQL_API_FN
  sqlmonss (
    unsigned long   version,
    _SQLOLDCHAR     *reserved,
    sqlma           *sqlma_ptr,
    unsigned long   buffer_length,
    void            *buffer_area,
    sqlm_collected  *collected,
    struct sqlca    *sqlca);
/* ... */
```

sqlmonss - Get Snapshot

Generic API Syntax

```
/* File: sqlmon.h */
/* API: Get Snapshot */
/* ... */
int SQL_API_FN
  sqlgmnss (
    unsigned long   reserved_lgth,
    struct   sqlca  *sqlca,
    sqlm_collected  *collected,
    void            *buffer_area,
    unsigned long   buffer_length,
    sqlma           *sqlma_ptr,
    _SQLOLDCHAR     *reserved,
    unsigned long   version);
/* ... */
```

API Parameters

reserved_lgth
A 4-byte unsigned integer representing the length of the reserved area.

sqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

collected
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. For more information about this structure, see "SQLM-COLLECTED" on page 407.

buffer_area
Output. Pointer to the user-defined data area into which the snapshot information will be returned. For information about interpreting the data returned in this buffer, see the *System Monitor Guide and Reference*.

buffer_length
Input. The length of the data buffer. Use "sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer" on page 218 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.

sqlma_ptr
Input. Pointer to the user-allocated *sqlma* (monitor area) structure. This structure specifies the type(s) of data to be collected. For more information, see "SQLMA" on page 412.

reserved
Reserved for future use. Must be set to NULL.

version
: 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`

  If requesting data for a version higher than the current server, the database monitor only returns data for its level (see the *server_version* field in "SQLM-COLLECTED" on page 407).

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

Sample Programs

C	\sqllib\samples\c\dbsnap.c

Usage Notes

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

For detailed information about the use of the database monitor APIs, and for a summary of all database monitor data elements and monitoring groups, see the *System Monitor Guide and Reference*.

See Also

"sqlmon - Get/Update Monitor Switches" on page 212
"sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer" on page 218
"sqlmrset - Reset Monitor" on page 221.

sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer

Estimates the buffer size needed by “sqlmonss - Get Snapshot” on page 215.

Scope

This API only affects the instance to which the calling application is attached.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. 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

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Estimate Size Required for sqlmonss() Output Buffer */
/* ... */
int SQL_API_FN
  sqlmonsz (
    unsigned long  version,
    _SQLOLDCHAR    *reserved,
    sqlma          *sqlma_ptr,
    unsigned long  *buff_size,
    struct sqlca   *sqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlmon.h */
/* API: Estimate Size Required for sqlmonss() Output Buffer */
/* ... */
int SQL_API_FN
  sqlgmnsz (
    unsigned long   reserved_lgth,
    struct   sqlca  *sqlca,
    unsigned long   *buff_size,
    sqlma           *sqlma_ptr,
    _SQLOLDCHAR     *reserved,
    unsigned long   version);
/* ... */
```

API Parameters

reserved_lgth
Reserved for future use. This value should be set to zero.

sqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

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

sqlma_ptr
Input. Pointer to the user-allocated *sqlma* (monitor area) structure. This structure specifies the type(s) of snapshot data to be collected, and can be reused as input to "sqlmonss - Get Snapshot" on page 215. For more information about this structure, see "SQLMA" on page 412.

reserved
Reserved for future use. Must be set to NULL.

version
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`

If requesting data for a version higher than the current server, the database monitor only returns data for its level (see the *server_version* field in "SQLM-COLLECTED" on page 407).

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

Sample Programs

C	\sqllib\samples\c\monsz.sqc

Usage Notes

This function generates a significant amount of overhead. Allocating and freeing memory dynamically for each **sqlmonss** call is also expensive. If calling **sqlmonss** 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 **sqlmonsz**.

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 “sqlmonss - Get Snapshot” on page 215. If an error is returned by **sqlmonss** because of insufficient buffer space to hold the output, call this API again to determine the new size requirements.

For detailed information about the use of the database monitor APIs, and for a summary of all database monitor data elements and monitoring groups, see the *System Monitor Guide and Reference*.

See Also

“sqlmon - Get/Update Monitor Switches” on page 212
“sqlmonss - Get Snapshot” on page 215
“sqlmrset - Reset Monitor” on page 221.

sqlmrset - Reset Monitor

Resets the database system monitor data of a specified database, or of all active databases, for the application issuing the call.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

Instance. 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

sqlmon.h

C API Syntax

```
/* File: sqlmon.h */
/* API: Reset Monitor */
/* ... */
int SQL_API_FN
  sqlmrset (
    unsigned long  version,
    _SQLOLDCHAR    *reserved,
    unsigned long  reset_all,
    _SQLOLDCHAR    *db_alias,
    struct sqlca   *sqlca);
/* ... */
```

sqlmrset - Reset Monitor

Generic API Syntax

```
/* File: sqlmon.h */
/* API: Reset Monitor */
/* ... */
int SQL_API_FN
  sqlgmrst (
    unsigned short  dbnamel,
    unsigned long   reserved_lgth,
    struct   sqlca  *sqlca,
    _SQLOLDCHAR     *db_alias,
    unsigned long   reset_all,
    _SQLOLDCHAR     *reserved,
    unsigned long   version);
/* ... */
```

API Parameters

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

reserved_lgth
: Reserved for future use. Users should set this value to zero.

sqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

db_alias
: Input. The name that is used to reference the database.

  If `SQLM_ON` is specified for the *reset_all* parameter, this alias is ignored, and the data areas for all active databases are reset.

reset_all
: Input. Indicates whether to reset data areas for a specific database, or for all active databases. Set this parameter to one of the following (defined in `sqlmon`):

  SQLM_OFF
  : Resets data areas for a specific database.

  SQLM_ON
  : Resets data areas for all active databases.

reserved
: Reserved for future use. Must be set to NULL.

version
: 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`

If requesting data for a version higher than the current server, the database monitor only returns data for its level (see the *server_version* field in “SQLM-COLLECTED” on page 407).

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

Sample Programs

C	\sqllib\samples\c\monreset.c

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 (see “sqlfxsys - Get Database Manager Configuration” on page 204). These settings can be overridden with “sqlmon - Get/Update Monitor Switches” on page 212.

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 “sqlmon - Get/Update Monitor Switches” on page 212.

For detailed information about the use of the database monitor APIs, and for a summary of all database monitor data elements and monitoring groups, see the *System Monitor Guide and Reference*.

See Also

“sqlmon - Get/Update Monitor Switches” on page 212
“sqlmonss - Get Snapshot” on page 215
“sqlmonsz - Estimate Size Required for sqlmonss() Output Buffer” on page 218.

sqlogstt - Get SQLSTATE Message

Retrieves the message text associated with an SQLSTATE.

Authorization

None

Required Connection

None

API Include File

sql.h

C API Syntax

```
/* File: sql.h */
/* API: Get SQLSTATE Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlogstt (
    char * pBuffer,
    short BufferSize,
    short LineWidth,
    char * pSqlstate);
/* ... */
```

Generic API Syntax

```
/* File: sql.h */
/* API: Get SQLSTATE Message */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggstt (
    short BufferSize,
    short LineWidth,
    char * pSqlstate,
    char * pBuffer);
/* ... */
```

API Parameters

BufferSize

Input. Size, in bytes, of a string buffer to hold the retrieved message text.

LineWidth
: Input. The maximum line width for each line of message text. Lines are broken on word boundaries. A value of zero indicates that the message text is returned without line breaks.

pSqlstate
: Input. A string containing the SQLSTATE for which the message text is to be retrieved. This field is alphanumeric and must be either five-digit (specific SQLSTATE) or two-digit (SQLSTATE class, first two digits of an SQLSTATE). This field does not need to be NULL-terminated if 5 digits are being passed in, but must be NULL-terminated if 2 digits are being passed.

pBuffer
: Output. A pointer to a string buffer where the message text is to be placed. If the message must be truncated to fit in the buffer, the truncation allows for the null string terminator character.

REXX API Syntax

```
GET MESSAGE FOR SQLSTATE sqlstate INTO :msg [LINEWIDTH width]
```

REXX API Parameters

sqlstate
: The SQLSTATE for which the message text is to be retrieved.

msg
: REXX variable into which the message is placed.

width
: Maximum line width for each line of the message text. The line is broken on word boundaries. If a value is not specified, or this parameter is set to 0, the message text returns without line breaks.

Sample Programs

COBOL	\sqllib\samples\cobol\checkerr.cbl
FORTRAN	\sqllib\samples\fortran\util.f

Usage Notes

One message is returned per call.

A LF/NULL sequence is placed at the end of each message.

If a positive line width is specified, LF/NULL sequences are inserted between words so that the lines do not exceed the line width.

If a word is longer than a line width, the line is filled with as many characters as will fit, a LF/NULL is inserted, and the remaining characters are placed on the next line.

Return Codes

Code	**Message**
+i	Positive integer indicating the number of bytes in the formatted message. If this is greater than the buffer size input by the caller, the message is truncated.
-1	Insufficient memory available for message formatting services to function. The requested message is not returned.
-2	The SQLSTATE is in the wrong format. It must be alphanumeric and be either 2 or 5 digits in length.
-3	Message file inaccessible or incorrect.
-4	Line width is less than zero.
-5	Invalid *sqlca*, bad buffer address, or bad buffer length.

If the return code is -1 or -3, the message buffer will contain further information about the problem.

See Also

"sqlaintp - Get Error Message" on page 15.

sqluadau - Get Authorizations

Reports the authorities of the current user from values found in the database manager configuration file and the authorization system catalog view (SYSCAT.DBAUTH).

Authorization

None

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Authorizations */
/* ... */
SQL_API_RC SQL_API_FN
  sqluadau (
    struct sql_authorizations * pAuthorizations,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Authorizations */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgadau (
    struct sql_authorizations * pAuthorizations,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

pAuthorizations

Input/Output. Pointer to the *sql_authorizations* structure. This array of short integers indicates which authorizations the current user holds. The first element in the structure, *sql_authorizations_len*, must be initialized to the size of the buffer being passed, prior to calling this API. For more information about the *sql_authorizations* structure, see “SQL-AUTHORIZATIONS” on page 340.

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
GET AUTHORIZATIONS :value
```

REXX API Parameters

value

A compound REXX host variable to which the authorization level is returned. In the following, XXX represents the host variable name. Values are `0` for no, and `1` for yes.

XXX.0	Number of elements in the variable (always 18)
XXX.1	Direct SYSADM authority
XXX.2	Direct DBADM authority
XXX.3	Direct CREATETAB authority
XXX.4	Direct BINDADD authority
XXX.5	Direct CONNECT authority
XXX.6	Indirect SYSADM authority
XXX.7	Indirect DBADM authority
XXX.8	Indirect CREATETAB authority
XXX.9	Indirect BINDADD authority
XXX.10	Indirect CONNECT authority
XXX.11	Direct SYSCTRL authority
XXX.12	Indirect SYSCTRL authority
XXX.13	Direct SYSMAINT authority
XXX.14	Indirect SYSMAINT authority
XXX.15	Direct CREATE_NOT_FENC authority
XXX.16	Indirect CREATE_NOT_FENC authority
XXX.17	Direct IMPLICIT_SCHEMA authority
XXX.18	Indirect IMPLICIT_SCHEMA authority.

Sample Programs

C	\sqllib\samples\c\dbauth.sqc
COBOL	\sqllib\samples\cobol\dbauth.sqb
FORTRAN	\sqllib\samples\fortran\dbauth.sqf
REXX	\sqllib\samples\rexx\dbauth.cmd

Usage Notes

Direct authorities are acquired by explicit commands that grant the authorities to a user ID. Indirect authorities are based on authorities acquired by the groups to which a user belongs.

Note: PUBLIC is a special group to which all users belong.

If there are no errors, each element of the *sql_authorizations* structure contains a `0` or a `1`. A value of `1` indicates that the user holds that authorization; `0` indicates that the user does not.

sqlubkp - Backup Database

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

Scope

This API only affects the node on which it is executed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

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

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Backup Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlubkp (
    char * pDbAlias,
    unsigned long BufferSize,
    unsigned long BackupMode,
    unsigned long BackupType,
    unsigned long CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    unsigned long NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaTargetList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    unsigned long VendorOptionsSize,
    void * pVendorOptions,
    unsigned long Parallelism,
    unsigned long * pBackupSize,
    void * pReserved4,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Backup Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgbkp (
    unsigned short DbAliasLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    unsigned short * pReserved1,
    char * pDbAlias,
    unsigned long BufferSize,
    unsigned long BackupMode,
    unsigned long BackupType,
    unsigned long CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    unsigned long NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaTargetList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    unsigned long VendorOptionsSize,
    void * pVendorOptions,
    unsigned long Parallelism,
    unsigned long * pBackupSize,
    void * pReserved4,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

UserNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
: Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

pReserved1.
: Reserved for future use.

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

BufferSize
: Input. Backup buffer size in allocation units of 4KB. Minimum is 16 units.

BackupMode
: Input. Specifies the backup mode. Valid values (defined in `sqlutil`) are:

  SQLUB_OFFLINE
  : Offline gives an exclusive connection to the database.

  SQLUB_ONLINE
  : Online allows database access by other applications while the backup occurs.

BackupType
: Input. Specifies the type of backup to be taken. Valid values (defined in `sqlutil`) are:

  SQLUB_FULL
  : Full database backup.

  SQLUB_TABLESPACE
  : Table space level backup. For a table space level backup, provide a list of table spaces in the *pTablespaceList* parameter.

CallerAction
: Input. Specifies action to be taken. Valid values (defined in `sqlutil`) are:

  SQLUB_BACKUP
  : Start the backup.

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

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

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

  SQLUB_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.

  SQLUB_PARM_CHECK
  : Used to validate parameters without performing a backup.

pApplicationId
: Output. Supply a buffer of length SQLU_APPLID_LEN+1 (defined in `sqlutil`). The API will return a string identifying the agent servicing the

application. Can be used to obtain information about the progress of the backup operation using the database monitor.

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

NumBuffers
: Input. Specifies number of backup buffers to be used.

pTablespaceList
: Input. List of table spaces to be backed up. Required for table space level backup only. See structure "SQLU-TABLESPACE-BKRST-LIST" on page 423.

pMediaTargetList
: Input. This structure allows the caller to specify the destination for the backup. The information provided depends on the value of the *media_type* field. The valid values for *media_type* (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
: Local devices. Allows a combination of tapes, disks or diskettes. Provide a list of *sqlu_media_entry*. On OS/2 or the Windows operating system, the entries can be directory paths only, not tape device names.

SQLU_ADSM_MEDIA
: ADSM. No additional input is required. The ADSM shared library provided with DB2 is used. If a different version of the ADSM shared library is desired, use `SQLU_OTHER_MEDIA` and provide the shared library name.

SQLU_OTHER_MEDIA
: Vendor product. Provide the shared library name in an *sqlu_vendor* structure.

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

For more information, see structure "SQLU-MEDIA-LIST" on page 417, and the *Administration Guide*.

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

pPassword
: Input. A string containing the password to be used with the user name.

pReserved2
: Reserved for future use.

VendorOptionsSize
: Input. The length of the *pVendorOptions* field.

pVendorOptions
: 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.

Parallelism
: Input. Degree of parallelism (number of buffer manipulators).

pBackupSize
: Output. Size of the backup image (in MB). Can be set to NULL.

pReserved4
: Reserved for future use.

pReserved3
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
BACKUP DATABASE dbalias USING :value [USER username USING password]

[TABLESPACE :tablespacenames] [ONLINE]

[LOAD vendor-library [OPTIONS vendor-options] [OPEN num-sessions SESSIONS] |
 TO :target-area |
 USE ADSM [OPEN num-sessions SESSIONS] |
 USER_EXIT]

[ACTION caller-action] [WITH num-buffers BUFFERS] [BUFFERSIZE buffer-size]
[PARALLELISM parallelism-degree]
```

REXX API Parameters

dbalias
: Alias of the database to be backed up.

value
: A compound REXX host variable to which the database backup information is returned. In the following, XXX represents the host variable name:

XXX.0	Number of elements in the variables (always 2)
XXX.1	The timestamp of the backup image
XXX.2	An application ID that identifies the agent that serves the application.

username
: Identifies the user name under which to back up the database.

password
: The password used to authenticate the user name.

tablespacenames
: A compound REXX host variable containing a list of table spaces to be backed up. In the following, XXX is the name of the host variable:

XXX.0	Number of table spaces to be backed up
XXX.1	First table space name

XXX.2	Second table space name
XXX.3	and so on.

vendor-library
The name of the shared library (DLL on OS/2 or the Windows operating system) containing the vendor backup and restore I/O functions to be used. It may contain the full path. If the full path is not given, defaults to the path on which the user exit program resides.

vendor-options
Information required by the vendor functions.

num-sessions
The number of I/O sessions to be used with ADSM or the vendor product.

target-area
Local devices. Allows a combination of tapes, disks or diskettes. Provide a list in "SQLU-MEDIA-LIST" on page 417. On OS/2 or the Windows operating system, the entries can be directory paths only, not tape device names.

caller-action
Specifies action to be taken. Valid values are:

SQLUB_BACKUP
Start the backup.

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

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

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

SQLUB_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.

SQLUB_PARM_CHECK
Used to validate parameters without performing a backup.

num-buffers
Number of backup buffers to be used.

buffer-size
Backup buffer size in allocation units of 4KB. Minimum is 16 units.

parallelism-degree
Number of buffer manipulators.

Sample Programs

C	\sqllib\samples\c\backrest.c
COBOL	\sqllib\samples\cobol\backrest.cbl
FORTRAN	\sqllib\samples\fortran\backrest.f

Usage Notes

For information about database level backup, table space level backup, online and offline backup, backup file names, and supported devices, see the *Command Reference*.

For a general discussion of backup, see “Recovering a Database” in the *Administration Guide*.

See Also

“sqlemgdb - Migrate Database” on page 145
“sqluroll - Rollforward Database” on page 300
“sqlurst - Restore Database” on page 309.

sqludrdt - Redistribute Nodegroup

Redistributes data across the nodes in a nodegroup. The current data distribution, whether it is uniform or skewed, can be specified. The redistribution algorithm selects the partitions to be moved based on the current data distribution.

This API can only be called from the catalog node. Use the LIST DATABASE DIRECTORY command (see the *Command Reference*) to determine which node is the catalog node for each database.

Scope

This API affects all nodes in the nodegroup.

Authorization

One of the following:

sysadm
sysctrl
dbadm

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Redistribute Nodegroup */
/* ... */
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);
/* ... */
```

sqludrdt - Redistribute Nodegroup

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Redistribute Nodegroup */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgdrdt (
    unsigned short NodeGroupNameLen,
    unsigned short TargetPMapFileNameLen,
    unsigned short DataDistFileNameLen,
    char * pNodeGroupName,
    char * pTargetPMapFileName,
    char * pDataDistFileName,
    SQL_PDB_NODE_TYPE * pAddList,
    unsigned short AddCount,
    SQL_PDB_NODE_TYPE * pDropList,
    unsigned short DropCount,
    unsigned char DataRedistOption,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

NodeGroupNameLen
: The length of the name of the nodegroup.

TargetPMapFileNameLen
: The length of the name of the target partitioning map file.

DataDistFileNameLen
: The length of the name of the data distribution file.

pNodeGroupName
: The name of the nodegroup to be redistributed.

pTargetPMapFileName
: The name of the file that contains the target partitioning map. If a directory path is not specified as part of the file name, the current directory is used. This parameter is used when the *DataRedistOption* value is `T`. The file should be in character format and contain either 4 096 entries (for a multi-node nodegroup) or 1 entry (for a single-node nodegroup). Entries in the file indicate node numbers. Entries can be in free format.

pDataDistFileName
: The name of the file that contains input distribution information. If a directory path is not specified as part of the file name, the current directory is used. This parameter is used when the *DataRedistOption* value is `U`. The file should be in character format and contain 4 096 positive integer entries. Each entry in the file should indicate the weight of the corresponding partition. The sum of the 4 096 values should be less than or equal to 4 294 967 295.

pAddList
The list of nodes to add to the nodegroup during the data redistribution. Entries in the list must be in the form: SQL_PDB_NODE_TYPE.

AddCount
The number of nodes to add to the nodegroup.

pDropList
The list of nodes to drop from the nodegroup during the data redistribution. Entries in the list must be in the form: SQL_PDB_NODE_TYPE.

DropCount
The number of nodes to drop from the nodegroup.

DataRedistOption
A single character that indicates the type of data redistribution to be done. Possible values are:

U
Specifies to redistribute the nodegroup to achieve a balanced distribution. If *pDataDistFileName* is null, the current data distribution is assumed to be uniform (that is, each hash partition represents the same amount of data). If *pDataDistFileName* is not null, the values in this file are assumed to represent the current data distribution. When the *DataRedistOption* is `U`, the *pTargetPMapFileName* should be null.

Nodes specified in the add list are added, and nodes specified in the drop list are dropped from the nodegroup.

T
Specifies to redistribute the nodegroup using *pTargetPMapFileName*. For this option, *pDataDistFileName*, *pAddList*, and *pDropList* should be null, and both *AddCount* and *DropCount* must be zero.

C
Specifies to continue a redistribution operation that failed. For this option, *pTargetPMapFileName*, *pDataDistFileName*, *pAddList*, and *pDropList* should be null, and both *AddCount* and *DropCount* must be zero.

R
Specifies to roll back a redistribution operation that failed. For this option, *pTargetPMapFileName*, *pDataDistFileName*, *pAddList*, and *pDropList* should be null, and both *AddCount* and *DropCount* must be zero.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Usage Notes

When a redistribution operation is done, a message file is written to the `$HOME/sqllib/redist` directory. The file name has the following format:

```
database-name.nodegroup-name.timestamp
```

The time stamp value is the time at which the API was called.

This utility performs intermittent COMMITs during processing.

Use the ALTER NODEGROUP statement to add nodes to a nodegroup. This statement permits one to define the containers for the table spaces associated with the nodegroup. See the *SQL Reference* for details.

Note: DB2 Parallel Edition for AIX Version 1 syntax, with ADD NODE and DROP NODE options, is supported for users with *sysadm* or *sysctrl* authority. For ADD NODE, containers are created like the containers on the lowest node number of the existing nodes within the nodegroup.

All packages having a dependancy on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute nodegroup 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 “sqlustat - Runstats” on page 319 after the redistribute nodegroup operation has completed.

See Also

“sqlarbnd - Rebind” on page 23.

sqluexpr - Export

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement.

Authorization

One of the following:

sysadm
dbadm

or CONTROL or SELECT privilege on each participating table or view.

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Export */
/* ... */
SQL_API_RC SQL_API_FN
  sqluexpr (
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    sqlu_media_list * pLobFileList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluexpt_out*  pOutputInfo,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

sqluexpr - Export

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Export */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgexpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    sqlu_media_list * pLobFileList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluexpt_out*  pOutputInfo,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

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

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

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

pLobPathList
Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files are to be stored.

When file space is exhausted on the first path in this list, the API will use the second path, and so on.

pLobFileList
Input. An *sqlu_media_list* using *media_type* `SQLU_CLIENT_LOCATION`, and the *sqlu_location_entry* structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on.

When creating LOB files during an export, file names are constructed by appending the current base name from this list to the current path (from *pLobFilePath*), and then appending a 3-digit sequence number. For example, if the current LOB path is the directory `/u/foo/lob/path`, and the current LOB file name is `bar`, then the LOB files created will be `/u/foo/lob/path/bar.001`, `/u/foo/lob/pah/bar.002`, and so on.

pDataDescriptor
: Input. Pointer to an *sqldcol* structure specifying the column names for the output file. The value of the *dcolmeth* field determines how the remainder of the information provided in this parameter is interpreted by EXPORT. Valid values for this field during an EXPORT (defined in `sqlutil`) are:

SQL_METH_N
Names

SQL_METH_D
Default.

If *dcolmeth* is `SQL_METH_N`, specified names are given for the columns in the external file.

If *pDataDescriptor* is NULL, or *dcolmeth* is set to `SQL_METH_D`, default names are used for the columns in the external file. In this case, the number of columns and the column specification array are both ignored. The column names in the external file are derived from the processing of the SELECT statement specified in *pActionString*.

For more information, see “SQLDCOL” on page 361.

pActionString
: Input. Pointer to a structure containing a valid dynamic SQL SELECT statement. The structure contains a 2-byte length field, followed by the characters that make up the SELECT statement. The SELECT statement specifies the data to be extracted from the database and written to the external file.

The columns for the external file (from *pDataDescriptor*), and the database columns from the SELECT statement, are matched according to their respective list/structure positions. The first column of data selected from the database is placed in the first column of the external file, and its column name is taken from the first element of the external column array.

pFileType
: Input. A string indicating the format of the data within the external file. Supported external file formats (defined in `sqlutil`) are:

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

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

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

pFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

For more information, see the *Command Reference*.

pMsgFileName
Input. A string containing the destination for EXPORT error, warning, and informational messages. Can be the path and name of an operating system file or a standard device. If the file already exists, it is overwritten. If it does not exist, a file is created.

CallerAction
Input. The action requested by the caller. It is defined as an integer by the application. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some service prior to completing the requested export, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. The action requested by the utility has completed, so the system can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. The requested action was not performed, so the system terminates the initial request.

pOutputInfo
Ouput. Return value of the number of rows exported. For more information about this structure, see “SQLUEXPT-OUT” on page 425.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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 within the data file. The supported file formats are:

DEL	Delimited ASCII
WSF	Worksheet formats
IXF	PC version of Integrated Exchange Format.

filetmod
A host variable containing additional information unique to the chosen file type. If no MODIFIED BY clause is specified, the default *filetmod* is used.

dcoldata
A compound REXX host variable containing the alternate column names to be used in the data file. In the following, XXX is 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.

msgfile
File (or 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.

Sample Programs

C	\sqllib\samples\c\impexp.sqc
COBOL	\sqllib\samples\cobol\impexp.sqb
FORTRAN	\sqllib\samples\fortran\impexp.sqf
REXX	\sqllib\samples\rexx\impexp.cmd

Usage Notes

Be sure to complete all table operations and release all locks before calling the EXPORT API. This can be done either by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK. A COMMIT is performed during the export process.

A warning message is issued if the number of columns (*dcolnum*) in the external column name array, *pDataDescriptor*, is not equal to the number of columns generated by the SELECT statement. In this case, the number of columns written to the external file is the lesser of the two numbers. Excess database columns or external column names are not used to generate the output file.

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 `db2uexpm.bnd` module or any other shipped `.bnd` files are bound manually, the **format** option on the binder must not be used.

The EXPORT utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.

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 (moving, for example, between OS/2 and AIX systems), fields containing the row separators will shrink or expand.

PC/IXF file format specifications permit migration of data between OS/2 (IBM Extended Services for OS/2, OS/2 Extended Edition and DB2 for OS/2) databases and DB2 for AIX databases via export, binary copying of files between OS/2 and AIX, and import. The file copying step is not necessary if the source and the target databases are both accessible from the same client.

DB2 Connect can be used to export tables from DRDA servers such as DB2 for MVS, SQL/DS, and OS/400. Only PC/IXF export is supported.

The EXPORT utility will not create multiple-part PC/IXF files when invoked from an AIX system.

Index definitions for a table are included in the PC/IXF file when the contents of a single database table are exported to a PC/IXF file with a *pActionString* beginning with `SELECT * FROM tablename`, and the *pDataDescriptor* parameter specifying default names. Indexes are not saved for views, or if the SELECT clause of the *pActionString* includes a join. A WHERE clause, a GROUP BY clause, or a HAVING clause in the *pActionString* will not prevent the saving of indexes.

Export 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`.

See Also

"sqluimpr - Import" on page 271
"sqluload - Load" on page 282.

sqlugrpn - Get Row Partitioning Number

Returns the partition number and the node number based on the partitioning key values. An application can use this information to determine at which node a specific row of a table is stored.

The partitioning data structure, “SQLUPI” on page 439, is the input for this API. The structure can be returned by “sqlugtpi - Get Table Partitioning Information” on page 252. Another input is the character representations of the corresponding partitioning key values. The output is a partition number generated by the partitioning strategy and the corresponding node number from the partitioning map. If the partitioning map information is not provided, only the partition number is returned. This can be useful when analyzing data distribution.

The database manager does not need to be running when this API is called.

Scope

This API can be invoked from any node in the `db2nodes.cfg` file.

Authorization

None

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Row Partitioning Number */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugrpn (
    unsigned short num_ptrs,
    unsigned char ** ptr_array,
    unsigned short * ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi * part_info,
    short * part_num,
    SQL_PDB_NODE_TYPE * node_num,
    unsigned short chklvl,
    struct sqlca * sqlca,
    short dataformat,
    void * pReserved1,
    void * pReserved2);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Row Partitioning Number */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggrpn (
    unsigned short num_ptrs,
    unsigned char ** ptr_array,
    unsigned short * ptr_lens,
    unsigned short ctrycode,
    unsigned short codepage,
    struct sqlupi * part_info,
    short * part_num,
    SQL_PDB_NODE_TYPE * node_num,
    unsigned short chklvl,
    struct sqlca * sqlca,
    short dataformat,
    void * pReserved1,
    void * pReserved2);
/* ... */
```

API Parameters

num_ptrs

The number of pointers in *ptr_array*. The value must be the same as the one specified for *part_info*; that is, *part_info->sqld*.

ptr_array

An array of pointers that points to the character representations of the corresponding values of each part of the partitioning key specified in *part_info*. If a null value is required, the corresponding pointer is set to null.

ptr_lens

An array of unsigned integers that contains the lengths of the character representations of the corresponding values of each part of the partitioning key specified in *part_info*.

ctrycode

The country code of the target database. For a list of valid country code values, see one of the *Quick Beginnings* books.

This value can also be obtained from the database configuration file (see the GET DATABASE CONFIGURATION command in the *Command Reference*.

codepage

The code page of the target database. For a list of valid code page values, see one of the *Quick Beginnings* books.

This value can also be obtained from the database configuration file (see the GET DATABASE CONFIGURATION command in the *Command Reference*.

part_info
: A pointer to the *sqlupi* structure. For more information about this structure, see “SQLUPI” on page 439.

part_num
: A pointer to a 2-byte signed integer that is used to store the partition number.

node_num
: A pointer to an SQL_PDB_NODE_TYPE field used to store the node number. If the pointer is null, no node number is returned.

chklvl
: An unsigned integer that specifies the level of checking that is done on input parameters. If the value specified is zero, no checking is done. If any non-zero value is specified, all input parameters are checked.

sqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

dataformat
: Specifies the representation of partitioning key values. Valid values are:

  SQL_CHARSTRING_FORMAT
  All partitioning key values are represented by character strings. This is the default value.

  SQL_PACKEDDECIMAL_FORMAT
  All decimal column partitioning key values are in packed decimal format.

  SQL_BINARYNUMERICS_FORMAT
  All numeric partitioning key values are in binary format.

pReserved1
: Reserved for future use.

pReserved2
: Reserved for future use.

Usage Notes

Datatypes supported on the operating system are the same as those that can be defined as a partitioning key.

CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC must be converted to the target code page before this API is called.

For numeric and datetime datatypes, the character representations must be at the code page of the respective system where the API is invoked.

If *node_num* is not NULL, the partitioning map must be supplied; that is, `part_info->pmaplen` is either 2 or 8192. Otherwise, SQLCODE -6038 is returned.

The partitioning key must be defined; that is, `part_info->sqld` must be greater than zero. Otherwise, SQLCODE -2032 is returned.

If a null value is assigned to a non-nullable partitioning column, SQLCODE -6039 is returned.

All the leading blanks and trailing blanks of the input character string are stripped, except for the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC datatypes, where only trailing blanks are stripped.

See Also

"sqlfxdb - Get Database Configuration" on page 201
"sqlugtpi - Get Table Partitioning Information" on page 252
"sqludrdt - Redistribute Nodegroup" on page 237.

sqlugtpi - Get Table Partitioning Information

Allows an application to obtain the partitioning information for a table. The partitioning information includes the partitioning map and the column definitions of the partitioning key. Information returned by this API can be passed to “sqlugrpn - Get Row Partitioning Number” on page 248 to determine the partition number and the node number for any row in the table.

To use this API, the application must be connected to the database that contains the table for which partitioning information is being requested.

Scope

This API can be executed on any node defined in the `db2nodes.cfg` file.

Authorization

For the table being referenced, a user must have at least one of the following:

sysadm authority
dbadm authority
CONTROL privilege
SELECT privilege

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Table Partitioning Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqlugtpi (
    unsigned char * tablename,
    struct sqlupi * part_info,
    struct sqlca * sqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Table Partitioning Information */
/* ... */
SQL_API_RC SQL_API_FN
  sqlggtpi (
    unsigned short tn_length,
    unsigned char * tablename,
    struct sqlupi * part_info,
    struct sqlca * sqlca);
/* ... */
```

API Parameters

tn_length
A 2-byte unsigned integer with the length of the table name.

tablename
The fully qualified name of the table.

part_info
A pointer to the *sqlupi* structure. For more information about this structure, see “SQLUPI” on page 439.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

See Also

“sqlugrpn - Get Row Partitioning Number” on page 248
“sqludrdt - Redistribute Nodegroup” on page 237.

sqluhcls - Close Recovery History File Scan

Ends a recovery history file scan and frees DB2 resources required for the scan. This API must be preceded by a successful call to “sqluhops - Open Recovery History File Scan” on page 259.

Authorization

None

Required Connection

Instance. It is not necessary to call ATTACH before issuing this API.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Close Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqluhcls (
    unsigned short Handle,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Close Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlghcls (
    unsigned short Handle,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Handle
: Input. Contains the handle for scan access that was returned by “sqluhops - Open Recovery History File Scan” on page 259.

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

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

REXX API Parameters

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

Sample Programs

C	\sqllib\samples\c\rechist.c
COBOL	\sqllib\samples\cobol\rechist.cbl
FORTRAN	\sqllib\samples\fortran\rechist.f
REXX	\sqllib\samples\rexx\rechist.cmd

Usage Notes

For a detailed description of the use of the recovery history file APIs, see “sqluhops - Open Recovery History File Scan” on page 259.

See Also

“sqluhgne - Get Next Recovery History File Entry” on page 256
“sqluhops - Open Recovery History File Scan” on page 259
“sqluhprn - Prune Recovery History File” on page 264
“sqluhupd - Update Recovery History File” on page 267.

sqluhgne - Get Next Recovery History File Entry

Gets the next entry from the recovery history file. This API must be preceded by a successful call to “sqluhops - Open Recovery History File Scan” on page 259.

Authorization

None

Required Connection

Instance. It is not necessary to call ATTACH before issuing this API.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Get Next Recovery History File Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqluhgne (
    unsigned short Handle,
    void * pReserved,
    struct sqluhinfo * pHistoryInfo,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Get Next Recovery History File Entry */
/* ... */
SQL_API_RC SQL_API_FN
  sqlghgne (
    unsigned short Handle,
    void * pReserved,
    struct sqluhinfo * pHistoryInfo,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

Handle
: Input. Contains the handle for scan access that was returned by "sqluhops - Open Recovery History File Scan" on page 259.

pReserved
: Reserved for future use.

pHistoryInfo
: Output. A pointer to the recovery history file entry information buffer (see "SQLUHINFO" on page 426). The history file information is returned in the memory pointed to by this parameter.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

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

REXX API Parameters

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

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

XXX.0	Number of first level elements in the variable (always 15)
XXX.1	Number of table space elements
XXX.2	Number of used table space elements
XXX.3	OPERATION (type of operation performed)
XXX.4	OBJECT (granularity of the operation)
XXX.5	OBJECT_PART (time stamp and sequence number)
XXX.6	OPTYPE (qualifier of the operation)
XXX.7	DEVICE_TYPE (type of device used)
XXX.8	FIRST_LOG (earliest log ID)
XXX.9	LAST_LOG (current log ID)
XXX.10	BACKUP_ID (identifier for the backup)
XXX.11	SCHEMA (qualifier for the table name)
XXX.12	TABLE_NAME (name of the unloaded/loaded table)

XXX.13.0	NUM_OF_TABLESPACES (number of table spaces involved in backup or restore)
XXX.13.1	Name of the first table space backed up/restored
XXX.13.2	Name of the second table space backed up/restored
XXX.13.3	and so on
XXX.14	LOCATION (where backup or copy is stored)
XXX.15	COMMENT (text to describe the entry).

Sample Programs

C	\sqllib\samples\c\rechist.c
COBOL	\sqllib\samples\cobol\rechist.cbl
FORTRAN	\sqllib\samples\fortran\rechist.f
REXX	\sqllib\samples\rexx\rechist.cmd

Usage Notes

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

For a detailed description of the use of the recovery history file APIs, see “sqluhops - Open Recovery History File Scan” on page 259.

See Also

“sqluhcls - Close Recovery History File Scan” on page 254
“sqluhops - Open Recovery History File Scan” on page 259
“sqluhprn - Prune Recovery History File” on page 264
“sqluhupd - Update Recovery History File” on page 267.

sqluhops - Open Recovery History File Scan

Starts a recovery history file scan.

Authorization

None

Required Connection

Instance. It is not necessary to call ATTACH before calling this API. If the database is cataloged as remote, an instance attachment to the remote node is established.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Open Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqluhops (
    char * pDbAlias,
    char * pTimestamp,
    char * pObjectName,
    unsigned short * pNumRows,
    unsigned short * pHandle,
    unsigned short CallerAction,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

sqluhops - Open Recovery History File Scan

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Open Recovery History File Scan */
/* ... */
SQL_API_RC SQL_API_FN
  sqlghops (
    unsigned short DbAliasLen,
    unsigned short TimestampLen,
    unsigned short ObjectNameLen,
    char * pDbAlias,
    char * pTimestamp,
    char * pObjectName,
    unsigned short * pNumRows,
    unsigned short * pHandle,
    unsigned short CallerAction,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

TimestampLen
Input. A 2-byte unsigned integer representing the length in bytes of the time stamp. Set to zero if no time stamp is provided.

ObjectNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the object name. Set to zero if no object name is provided.

pDbAlias
Input. A string containing the database alias.

pTimestamp
Input. 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.

pObjectName
Input. 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.

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

pHandle

Output. Upon return from the API, this parameter contains the handle for scan access. It is subsequently used in “sqluhgne - Get Next Recovery History File Entry” on page 256, and “sqluhcls - Close Recovery History File Scan” on page 254.

CallerAction

Input. Valid values (defined in `sqlutil`) are:

SQLUH_LIST_HISTORY

Select all of the records (backup, restore, load and unload) that pass the other filters.

SQLUH_LIST_BACKUP

Select only the backup and restore records that pass the other filters.

pReserved

Reserved for future use.

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

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

REXX API Parameters

database_alias

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

objname

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

timestamp

Specifies the time stamp to be used for selecting records. Records whose time stamp is equal to or greater than this value are selected. Setting this parameter to NULL prevents the filtering of entries using *timestamp*.

value

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

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

sqluhops - Open Recovery History File Scan

Sample Programs

C	\sqllib\samples\c\rechist.c
COBOL	\sqllib\samples\cobol\rechist.cbl
FORTRAN	\sqllib\samples\fortran\rechist.f
REXX	\sqllib\samples\rexx\rechist.cmd

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 loads and unloads, because this is the only information for tables in the history file.
- Specifying a table space will return records for backups, restores, loads, and unloads for the table space.

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

To list every entry in the history file, a typical application will perform the following steps:

1. Call **sqluhops**, which will return *pNumRows*
2. Allocate an *sqluhinfo* structure with space for *n tablespace* fields, where *n* is an arbitrary number
3. Set the *sqln* field of the *sqluhinfo* structure to *n*
4. In a loop, perform the following:
   - Call **sqluhgne** to fetch from the history file.
   - If **sqluhgne** returns an SQLCODE of `SQL_RC_OK`, use the *sqld* field of the *sqluhinfo* structure to determine the number of table space entries returned.
   - If **sqluhgne** returns an SQLCODE of `SQLUH_SQLUHINFO_VARS_WARNING`, not enough space has been allocated for all of the table spaces that DB2 is trying to return; free and reallocate the *sqluhinfo* structure with enough space for *sqld* table space entries, and set *sqln* to *sqld*.
   - If **sqluhgne** returns an SQLCODE of `SQLE_RC_NOMORE`, all recovery history files have been retrieved.
   - Any other SQLCODE indicates a problem.
5. When all of the information has been fetched, call "sqluhcls - Close Recovery History File Scan" on page 254 to free the resources allocated by the call to **sqluhops**.

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

See Also

“sqluhcls - Close Recovery History File Scan” on page 254
“sqluhgne - Get Next Recovery History File Entry” on page 256
“sqluhprn - Prune Recovery History File” on page 264
“sqluhupd - Update Recovery History File” on page 267.

sqluhprn - Prune Recovery History File

Deletes entries from the recovery history file.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database. To delete entries from the recovery 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

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Prune Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  sqluhprn (
    char * pTimestamp,
    unsigned short ForceOption,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Prune Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  sqlghprn (
    unsigned short TimestampLen,
    char * pTimestamp,
    unsigned short ForceOption,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

TimestampLen
Input. A 2-byte unsigned integer representing the length in bytes of the time stamp.

pTimestamp
Input. A string specifying the time stamp or part of a time stamp (minimum *yyyy*, or year) used to select records for deletion. All entries equal to or less than the time stamp will be deleted. A valid time stamp must be provided; there is no default behavior for a NULL parameter.

ForceOption
Input. Indicates whether history file entries corresponding to the most recent full backup and its restore set should be kept. The restore set includes all table space backups and load copies taken after the most recent full database backup. Valid values (defined in `sqlutil`) are:

SQLUH_NO_FORCE
The most recent restore set entries will be kept, even if the time stamp is less than or equal to the time stamp specified as input.

SQLUH_FORCE
The recovery 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.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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 recovery history file.

WITH FORCE OPTION
If specified, the recovery 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.

Sample Programs

C	\sqllib\samples\c\rechist.c
COBOL	\sqllib\samples\cobol\rechist.cbl
FORTRAN	\sqllib\samples\fortran\rechist.f
REXX	\sqllib\samples\rexx\rechist.cmd

Usage Notes

Pruning the recovery history file does not delete the actual backup, load, and unload files. The user must manually delete these files to free up the space they consume on storage media.

Attention:

If the latest full database backup is deleted from the media (in addition to being pruned from the recovery 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.

See Also

“sqluhcls - Close Recovery History File Scan” on page 254
“sqluhgne - Get Next Recovery History File Entry” on page 256
“sqluhops - Open Recovery History File Scan” on page 259
“sqluhupd - Update Recovery History File” on page 267.

sqluhupd - Update Recovery History File

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

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database. To update entries in the recovery 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

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Update Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  sqluhupd (
    char * pObjectPart,
    char * pNewLocation,
    char * pNewDeviceType,
    char * pNewComment,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

sqluhupd - Update Recovery History File

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Update Recovery History File */
/* ... */
SQL_API_RC SQL_API_FN
  sqlghupd (
    unsigned short ObjectPartLen,
    unsigned short NewLocationLen,
    unsigned short NewDeviceTypeLen,
    unsigned short NewCommentLen,
    char * pObjectPart,
    char * pNewLocation,
    char * pNewDeviceType,
    char * pNewComment,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

ObjectPartLen
: Input. A 2-byte unsigned integer specifying the length in bytes of the *pObjectPart* string.

NewLocationLen
: Input. A 2-byte unsigned integer specifying the length in bytes of the *pNewLocation* string. Set to zero if a new location is not provided.

NewDeviceTypeLen
: Input. A 2-byte unsigned integer specifying the length in bytes of the *pNewDeviceType* string. Set to zero if a new device type is not provided.

NewCommentLen
: Input. A 2-byte unsigned integer specifying the length in bytes of the *pNewComment* string. Set to zero if a new comment is not provided.

pObjectPart
: Input. A string specifying the identifier for the backup, restore, unload, or load copy image. This parameter has the form of a time stamp with a sequence number from 001 to 999.

pNewLocation
: Input. A string specifying a new location for the backup, restore, unload, or load copy image. Setting this parameter to NULL, or pointing to zero, leaves the value unchanged.

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

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

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

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

REXX API Parameters

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

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

Sample Programs

C	\sqllib\samples\c\rechist.c
COBOL	\sqllib\samples\cobol\rechist.cbl
FORTRAN	\sqllib\samples\fortran\rechist.f
REXX	\sqllib\samples\rexx\rechist.cmd

Usage Notes

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

The recovery history file is used for recording purposes only. It is not used directly by the restore or the roll-forward functions. During a restore, the location of the backup can be specified, and the history file is useful for tracking this location. The information can subsequently be provided to "sqlubkp - Backup Database" on page 230. Similarly, if the location of a load copy image is moved, roll-forward recovery must be informed of the new location and storage media. For additional details, see the *Administration Guide* and "sqluroll - Rollforward Database" on page 300.

See Also

"sqluhcls - Close Recovery History File Scan" on page 254
"sqluhgne - Get Next Recovery History File Entry" on page 256
"sqluhops - Open Recovery History File Scan" on page 259
"sqluhprn - Prune Recovery History File" on page 264.

sqluimpr - Import

Inserts data from an external file with a supported file format into a table or view. A faster alternative is “sqluload - Load” on page 282.

Authorization

- IMPORT using the INSERT option requires one of the following:

  sysadm
  dbadm
  CONTROL privilege on the table or view
  INSERT and SELECT privilege on the table or view.

- IMPORT to an existing table using the INSERT_UPDATE, REPLACE, or the REPLACE_CREATE option, requires one of the following:

  sysadm
  dbadm
  CONTROL privilege on the table or view.

- IMPORT to a table that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:

  sysadm
  dbadm
  CREATETAB authority on the database, and one of:
  - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema of the table exists.

Required Connection

Database

API Include File

sqlutil.h

sqluimpr - Import

C API Syntax

```
/* File: sqlutil.h */
/* API: Import */
/* ... */
SQL_API_RC SQL_API_FN
  sqluimpr (
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluimpt_in*  pImportInfoIn,
    struct sqluimpt_out*  pImportInfoOut,
    long * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Import */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgimpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluimpt_in*  pImportInfoIn,
    struct sqluimpt_out*  pImportInfoOut,
    long * NullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

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

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

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

pLobPathList
: Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA` and the *sqlu_media_entry* structure listing paths on the client where the LOB files can be found.

pDataDescriptor
: Input. Pointer to an *sqldcol* structure containing information about the columns being selected for import from the external file. The value of the *dcolmeth* field determines how the remainder of the information provided in this parameter is interpreted by IMPORT. Valid values for this field during an IMPORT (defined in `sqlutil`) are:

SQL_METH_N
: Names

SQL_METH_P
: Positions

SQL_METH_L
: Locations

SQL_METH_D
: Default.

If *dcolmeth* is `SQL_METH_N`, selection of columns from the external file is by name.

If *dcolmeth* is `SQL_METH_P`, selection of columns from the external file is by position.

If *dcolmeth* is `SQL_METH_L`, selection of columns from the external file is by 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 beginning/end 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. If *pDataDescriptor* is NULL, or is set to

`SQL_METH_D`, default selection of columns from the external file is done. In this case, the number of columns and the column specification array are both ignored. The first *n* columns of data in the external file are taken in their natural order, where *n* is the number of database columns into which the data is to be imported.

Anything that is not a valid specification of external columns, either by name, position, location, or default, is an error.

For more information, see "SQLDCOL" on page 361.

pActionString

Input. Pointer to a structure containing a 2-byte length field, followed by an array of characters. The array identifies the columns into which data is to be imported.

The character array is of the form:

```
{INSERT|INSERT_UPDATE|REPLACE|CREATE|REPLACE_CREATE}
INTO tname [(tcolumn-list)]
```

INSERT

The imported data is to be added to the data in the table, and the previously existing table data should not be changed.

INSERT_UPDATE

The imported rows are added for data with primary keys that are not in the table, and are updated for data with matching primary keys. This option is only valid when 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

The previously existing table data is deleted before the imported data is inserted into the table. The table definition and index definitions are not disturbed. (Indexes are deleted and replaced if `indexixf` is in *FileTypeMod*, and *FileType* is `SQL_IXF`.) If the table is not already defined, an error is returned.

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

CREATE

If the specified table name is not already defined, the table definition and the row contents are created using the PC/IXF information in the specified PC/IXF file. If the file was previously exported by the database manager, indexes are also created. If the specified table name is already defined, an error is returned. This option is valid for the PC/IXF file format only.

REPLACE_CREATE

If the specified table name is already defined, the table row contents are replaced using the PC/IXF row information in the PC/IXF file. If the table name is not already defined, the table definition and row contents are created using the PC/IXF information in the PC/IXF file. If the PC/IXF file was exported by the database manager, indexes are also created. This option is valid for the PC/IXF file format only.

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

Additional elements of the *pActionString* array are:

tname

The name of the table or view into which the data is to be inserted. Can use an alias for REPLACE, INSERT_UPDATE, or INSERT, except in the case of a down-level server, when a qualified or unqualified name should be used. If it is a view, it cannot be a read-only view.

tcolumn-list

A list of column names within the table or view into which the data is to be inserted. Commas must separate the list elements. If column names are not present, column names as defined in CREATE TABLE and ALTER TABLE statements are used.

The *tname* and the *tcolumn-list* correspond to the *tablename* and the *colname* list of SQL INSERT statements, and have the same restrictions.

The columns in *tcolumn-list* and the external columns (either specified or implied) are matched according to their position in the list or the structure (data from the first column specified in the *sqldcol* structure is inserted into the table or view field corresponding to the first element of the *tcolumn-list*).

If unequal numbers of columns are specified, the number of columns actually processed is the lesser of the two numbers. This could result in an error (because there are no values to place in some non-nullable table fields) or an informational message (because some external file columns are ignored).

pFileType

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

SQL_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

Nondelimited ASCII.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

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

pFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

For more information, see the *Command Reference*.

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

CallerAction
Input. The action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. *CallerAction* must be set to this value on the first call to the API.

If the initial call or any subsequent call returns and requires the caller to perform some action prior to completing the requested import, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. The action requested by the utility has completed, so the system can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. The action requested was not performed, so the system terminates the initial request.

pImportInfoIn
Input. An input structure. For information about this structure, see "SQLUIMPT-IN" on page 430.

pImportInfoOut
Output. An output structure. For information about this structure, see "SQLUIMPT-OUT" on page 431.

NullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. The number of elements in this array must match the number of columns in the input file; there is a one-to-one ordered correspondence between the elements of this array and the columns being imported from the data file. That is, the number of elements must equal the *dcolnum* field of the *pDataDescriptor* parameter. Each

element of the array contains a number identifying a column in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified column in the data file must contain a `Y` or an `N`. A `Y` indicates that the table column data is null, and `N` indicates that the table column data is not null.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

```
IMPORT FROM datafile OF filetype
[MODIFIED BY :filetmod]
[METHOD {L|N|P} USING :dcoldata]
[COMMITCOUNT :commitcnt] [RESTARTCOUNT :restartcnt]
MESSAGES msgfile
{INSERT|REPLACE|CREATE|INSERT_UPDATE|REPLACE_CREATE}
INTO tname [(:columns)]
[OUTPUT INTO :output]

CONTINUE IMPORT

STOP IMPORT
```

REXX API Parameters

datafile
Name of the file from which the data is to be imported.

filetype
The format of the data within the external file. The file formats supported are:

DEL Delimited ASCII

ASC Nondelimited ASCII

WSF Worksheet formats

IXF PC version of Integrated Exchange Format.

filetmod
A host variable containing additional information unique to the chosen file type. If no MODIFIED BY clause is specified, the default *filetmod* is used.

L|N|P
A character that indicates the method to be used to select columns within the external file. Valid values are:

L Location

N	Name
P	Position.

dcoldata

A compound REXX host variable containing information about the columns selected for import from the external file. The content of the structure depends upon the *method* selected. In the following description, XXX is the name of the host variable:

- Location method

XXX.0	Number of elements in the remainder of the host variable
XXX.1	A number representing the starting location of this column in the input file. This column is used as the first column in the database
XXX.2	A number representing the ending location of the column
XXX.3	A number representing the beginning location of this column in the input file. This column becomes the second column in the database
XXX.4	A number representing the ending location of the column
XXX.5	and so on.

- Name method

XXX.0	Number of column names contained within the host variable
XXX.1	First name
XXX.2	Second name
XXX.3	and so on.

- Position method

XXX.0	Number of column positions contained within the host variable
XXX.1	A column position in the external file
XXX.2	A column position in the external file
XXX.3	and so on.

tname

Name of the target table or view. Data cannot be imported to a read-only view.

columns

A REXX host variable containing the names of columns within the table or view into which the data is to be inserted. In the following, XXX is the name of the host variable:

XXX.0	Number of columns
XXX.1	First column name
XXX.2	Second column name
XXX.3	and so on.

msgfile
: File or device name where error and warning messages are sent. Path can be used for files.

commitcnt
: A host variable specifying that a COMMIT is to be performed after every *commitcnt* imported records.

restartcnt
: A host variable specifying that an import is to be started at record (*restartcnt*+1). The first *restartcnt* records are to be skipped.

output
: A compound REXX host variable into which information from the import will be passed. In the following, XXX is the name of the host variable:

XXX.1	Number of records read from the file during import
XXX.2	Number of records skipped before inserting or updating begins
XXX.3	Number of rows inserted into the target table
XXX.4	Number of rows of the target table updated with information from the imported records
XXX.5	Number of records that could not be imported
XXX.6	Number of records imported successfully and committed to the database, including rows inserted, rows updated, rows skipped, and rows rejected.

Sample Programs

C	\sqllib\samples\c\impexp.sqc
COBOL	\sqllib\samples\cobol\impexp.sqb
FORTRAN	\sqllib\samples\fortran\impexp.sqf
REXX	\sqllib\samples\rexx\impexp.cmd

Usage Notes

IMPORT 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.

IMPORT (in PC/IXF format) can be used to recover a previously exported table. The table returns to the state it was in when exported. This is distinct from the backup utility.

An INSERT, INSERT_UPDATE, REPLACE, or REPLACE_CREATE keyword in the parameter list controls whether the existing data in the table or view is deleted before the rows of imported data are added:

INSERT
: Inserts new rows, has no effect on existing rows.

INSERT_UPDATE
: Inserts new rows, and updates existing rows that have matching keys.

REPLACE
: Deletes all rows and repopulates the table.

REPLACE_CREATE
: If the table exists, deletes all rows and repopulates the table. If the table does not exist, creates and populates the table.

The caller action *repeat call* facility provides support for multiple PC/IXF files created on platforms that support diskettes.

Be sure to complete all table operations and release all locks before calling this API. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

When importing part of a file after a system failure, record the number of records imported every time a COMMIT is done. Whenever a COMMIT is performed, 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 after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

Importing IXF files to a remote database is much faster if the IXF file is on a hard drive rather than on diskettes. Non-default values for *pDataDescriptor*, or specifying an explicit list of table columns in the *pActionString*, makes importing to a remote database slower.

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 IMPORT is run against a remote database, and the output message file is very long (more than 60KB), the message file returned to the user on the client may be missing messages from the middle of the import. The first 30KB of message information and the last 30KB of message information are always retained.

After the old rows are deleted during a REPLACE or REPLACE_CREATE, the utility performs an automatic COMMIT. Consequently, if the system fails, or the application interrupts the database manager after the records are deleted, part or all of the old data is lost. Ensure that the old data is no longer needed before using these options.

When the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE, 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 partially filled data remains in the database. Use the REPLACE or the REPLACE_CREATE option to execute the whole import again, or use INSERT with the *restartcount* parameter set to the number of rows successfully imported.

By default, automatic commits are not done for the INSERT or the INSERT_UPDATE option. However, they are done if the *commitcnt* parameter is not zero. A full log results in a rollback.

IMPORT 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.

Data cannot be imported to a system table.

Views cannot be created with the IMPORT API.

One cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself).

To replace such a table or a view, do the following:

1. Drop all foreign keys in which the table is a parent.
2. Execute IMPORT.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating foreign keys, modify the data so that it will maintain referential integrity.

Referential constraints and key definitions are not preserved when creating tables using the PC/IXF file format.

See Also

"sqluexpr - Export" on page 241
"sqluload - Load" on page 282.

sqluload - Load

Loads data from files, tapes, or named pipes into a DB2 table.

Scope

This API only affects the node on which it is executed.

In a multi-node environment, this API can be used only with ASC or DEL files. IXF files can be loaded only if the table exists on a single node nodegroup.

Authorization

One of the following:

sysadm
dbadm

Required Connection

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

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  sqluload (
    sqlu_media_list * pDataFileList,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    short CallerAction,
    struct sqluload_in * pLoadInfoIn,
    struct sqluload_out * pLoadInfoOut,
    sqlu_media_list * pWorkDirectoryList,
    sqlu_media_list * pCopyTargetList,
    long * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

sqluload - Load

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgload (
    unsigned short FileTypeLen,
    unsigned short LocalMsgFileNameLen,
    unsigned short RemoteMsgFileNameLen,
    sqlu_media_list * pDataFileList,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    short CallerAction,
    struct sqluload_in * pLoadInfoIn,
    struct sqluload_out * pLoadInfoOut,
    sqlu_media_list * pWorkDirectoryList,
    sqlu_media_list * pCopyTargetList,
    long * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

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

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

pDataFileList
Input. A pointer to an *sqlu_media_list* structure used to provide a list of source files, devices, vendors or pipes.

The information provided in this structure depends on the value of the *media_type* field. Valid values (defined in `sqlutil`) are:

SQLU_SERVER_LOCATION
If the *media_type* field is set to this value, the caller provides information via *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_ADSM_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 ADSM sessions to initiate. LOAD 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. LOAD will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

Wherever a file name is provided, it should be fully qualified.

pLobPathList
Input. A pointer to an *sqlu_media_list* structure. For IXF, ASC, and DEL filetypes, 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/DEL files, and are appended to the paths provided.

The information provided in this structure depends on the value of the *media_type* field. Valid values (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
If set to this value, the caller provides information via *sqlu_media_entry* structures. The *sessions* field indicates the number of *sqlu_media_entry* structures provided.

SQLU_ADSM_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 ADSM sessions to initiate. LOAD 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. LOAD will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

pDataDescriptor
Input. Pointer to an *sqldcol* structure containing information about the columns being selected for loading from the external file.

If the *pFileType* parameter is set to `SQL_ASC`, the *dcolmeth* field of this structure must be `SQL_METH_L`. The user indicates the start and end locations for each column to be loaded.

If the file type is `SQL_DEL`, *dcolmeth* can be either `SQL_METH_P` or `SQL_METH_D`. If it is `SQL_METH_P`, the user must provide the column position from which the data comes. If it is `SQL_METH_D`, the first column in the file will be loaded into the first column of the table, and so on.

If the file type is `SQL_IXF`, *dcolmeth* can be one of `SQL_METH_P`, `SQL_METH_D`, or `SQL_METH_N`. The rules for DEL files apply here, except that `SQL_METH_N` indicates that file column names are to be provided in the *sqldcol* structure.

For more information, see "SQLDCOL" on page 361.

pActionString

Input. Specifies an action that affects the table. Pointer to an *sqlchar* structure that contains the following string:

```
"INSERT|REPLACE|RESTART|TERMINATE
into tbname [(column_list)][FOR EXCEPTION e_tbname]"
```

INSERT

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

REPLACE

Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed.

RESTART

Restarts LOAD after a previous load was interrupted.

It is important to keep track of the last commit point. This information is stored in the message file and is passed to LOAD. Use "sqluqry - Load Query" on page 291 to get this information if the database connection was lost during the load.

TERMINATE

Terminates a previously interrupted load and moves the table spaces in which the table resides from load pending state to recovery pending state. The table spaces cannot be used until a backup has been restored and the table spaces have been rolled forward. A restart should be issued before attempting to complete an interrupted load.

Note: This option is not recommended for general use; it should only be selected if an unrecoverable error has occurred.

into tbname

Specifies the table within the database into which the data is to be loaded. The table cannot be a system catalog 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 authorization ID.

(column_list)
Specifies the columns within the table 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_tbname
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied.

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

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII.

SQL_IXF
IXF (integrated exchange format, PC version) exported from the same or from another DB2 table.

For more information about file formats, see the *Command Reference*.

pFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

For more information, see the *Command Reference*.

pLocalMsgFileName
Input. A string containing the local file name to be used for output messages.

pRemoteMsgFileName
Input. A string containing the base name to be used on the server for temporary files. Temporary files are created to store messages, consistency points, and to delete phase information. Different extensions will be appended to this name for the various files. For more information about remote files, see page 289).

CallerAction
Input. Specifies an action that affects the utility. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. Must be set to this value or to SQLU_NOINTERRUPT for the first call.

SQLU_CONTINUE
Continue processing. The action requested by the utility has completed, so the system can continue processing the request. This option could be specified, for example, after a tape has been changed.

SQLU_TERMINATE
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in RECOVER_PENDING and QUIESCE_EXCLUSIVE state.

SQLU_NOINTERRUPT
Initial call. Do not suspend processing. Must be set to this value or to SQLU_INITIAL for the first call.

SQLU_ABORT
Abort 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.

pLoadInfoIn
Input. Optional pointer to the *sqluload_in* structure containing additional input parameters. See "SQLULOAD-IN" on page 433.

pLoadInfoOut
Output. Optional pointer to the *sqluload_out* structure containing additional output parameters. See "SQLULOAD-OUT" on page 437.

pWorkDirectoryList
Input. Optional work directories used for sorting index keys. If not provided, the `sqllib/tmp` directory is used.

pCopyTargetList
Input. If a copy image is to be created, this parameter contains target paths, devices, or a shared library to which the copy image is to be written.

The values provided in this structure depend on the value of the *media_type* field. Valid values for this field (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
If the copy is to be written to local media, set the *media_type* to this value and provide information about the targets in *sqlu_media_entry* structures. The *sessions* field specifies the number of *sqlu_media_entry* structures provided.

SQLU_ADSM_MEDIA
If the copy is to be written to ADSM, 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. LOAD will start the sessions with different sequence numbers, but with the same data provided in the one *sqlu_vendor* entry.

pNullIndicators
: Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. There is a one-to-one ordered correspondence between the elements of this array and the columns being loaded from the data file. That is, the number of elements must equal the *dcolnum* field of the *pDataDescriptor* parameter. Each element of the array contains a number identifying a location in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified location in the data file must contain a `Y` or an `N`. A `Y` indicates that the table column data is null, and `N` indicates that the table column data is not null.

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See "How the API Descriptions are Organized" on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\tload.sqc
COBOL	\sqllib\samples\cobol\tload.sqb
FORTRAN	\sqllib\samples\fortran\tload.sqf

Usage Notes

Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted.

The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not perform referential integrity or constraint checking. If these are included in the table definition, the tables are placed in check pending state, and the user must either force the check flag, or execute the SET CONSTRAINTS statement.

Remote Files

Remote file is a base file name to which DB2 appends different extensions to create files used by other functions (for example, `.msg` for **sqluqry**).

The remote file resides on the server machine and is accessed by the DB2 instance exclusively. Therefore, it is imperative that any file name qualification given to this parameter reflects the directory structure of the server, not the client, and that the DB2 instance owner has read and write permission on this file. In addition, the user must ensure that two loads are not issued that have the same fully-qualified remote file name.

There are several ways that the remote file name can be selected and qualified when the user has just given a partially qualified name, or no name at all:

- No remote file name is given in a load operation where the user is on the same machine as the database instance. In this case, the load utility will use the name *db2utmp* and qualify it with the current working directory of the user. Two loads from the same directory with this option will clash on the use of the remote file name, therefore this option is not recommended.
- No remote file name is given in a load operation, where the user is on a different machine than the database instance. In this case, the load utility will generate a name that will reside in the database directory. This effectively prevents the user from using the load query facility, since it requires the name of the remote file. In addition, the file name generated is not guaranteed to be unique, and therefore clashes may occur between different load operations. Therefore this option is not recommended.
- A non-fully-qualified file name is given in a load operation, where the user is on the same machine as the database instance. In this case the name is qualified by using the current directory of the user. The user must ensure that two loads are not issued from the same directory with the same remote file name.
- A non-fully-qualified file name is given in a load operation, where the user is on a different machine than the database instance. In this case the load utility will reject the file name. It must be fully qualified from the client.
- A fully-qualified file name is given in a load operation. This will be the file name used. The user must ensure that two loads are not issued with the same remote file name. This is the recommended usage.

Note: In an MPP system, the remote file must reside on a local disk, not on an NFS mount. If the file is on an NFS mount, there will be a significant performance decrement during the load operation.

See Also

“sqluqry - Load Query” on page 291
“sqluvqdp - Quiesce Tablespaces for Table” on page 324.

sqluqry - Load Query

Queries the server as to the status of the load.

Authorization

None

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Load Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqluqry (
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Load Query */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgqry (
    unsigned short LocalMsgFileNameLen,
    unsigned short RemoteMsgFileNameLen,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

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

pLocalMsgFileName
: Input. A string containing the name of the local file to be used for output messages.

pRemoteMsgFileName
: Input. A string containing the base name to be used on the server for temporary files of a load currently in progress.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See “How the API Descriptions are Organized” on page 8, or the *Embedded SQL Programming Guide*. For a description of the syntax, see the *Command Reference*.

Sample Programs

C	\sqllib\samples\c\qload.sqc
COBOL	\sqllib\samples\cobol\qload.sqb
FORTRAN	\sqllib\samples\fortran\qload.sqf

Usage Notes

This API reads the status of the load from the file specified by *pRemoteMsgFileName* and places the results in the file specified by *pLocalMsgFileName*. The remote file specified will be the same as the remote file specified on the call to the LOAD API.

sqlureot - Reorganize Table

Reorganizes a table by reconstructing the rows to eliminate fragmented data, and by compacting information.

Scope

This API affects all nodes in the nodegroup.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm
CONTROL privilege on the table.

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Reorganize Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqlureot (
    _SQLOLDCHAR * pTableName,
    _SQLOLDCHAR * pIndexName,
    _SQLOLDCHAR * pTablespace,
    struct sqlca * pSqlca);
/* ... */
```

sqlureot - Reorganize Table

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Reorganize Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgreot (
    unsigned short TablespaceLen,
    unsigned short IndexNameLen,
    unsigned short TableNameLen,
    struct sqlca * pSqlca,
    _SQLOLDCHAR * pTablespace,
    _SQLOLDCHAR * pIndexName,
    _SQLOLDCHAR * pTableName);
/* ... */
```

API Parameters

TablespaceLen
: Input. A 2-byte unsigned integer representing the length in bytes of the table space string. Set to zero if no table space is specified.

IndexNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the index name. Set to zero if no index is specified.

TableNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the table name.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

pTablespace
: Input. A string containing the name of the temporary table space if the caller wants a secondary work area when reorganizing a table. May be NULL.

pIndexName
: Input. The fully qualified index name to be used when reorganizing the user table. The records in the reorganized table are physically ordered according to this index. Setting this parameter to NULL causes the data to be reorganized in no specific order.

pTableName
: Input. Name of the table to be reorganized. Can be an alias, except in the case of a down-level server, when the fully qualified name of the table must be used.

REXX API Syntax

```
REORG TABLE tablename [INDEX iname] [USE tablespace_id]
```

REXX API Parameters

tablename
: The fully qualified name of the table.

iname
: The fully qualified index name used to reorganize the table. If an index name is not specified, the data is reorganized in no specific order.

tablespace_id
: The name of a temporary table space.

Sample Programs

C	\sqllib\samples\c\dbstat.sqc
COBOL	\sqllib\samples\cobol\dbstat.sqb
FORTRAN	\sqllib\samples\fortran\dbstat.sqf
REXX	\sqllib\samples\rexx\dbstat.cmd

Usage Notes

Tables that have been modified so many times that data is fragmented and access performance is noticeably slow are candidates for reorganization. Use "REORGCHK" in the *Command Reference* to determine whether a table needs reorganizing. Be sure to complete all database operations and release all locks before calling REORGANIZE TABLE. This may be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK. After reorganizing a table, use “sqlustat - Runstats” on page 319 to update the table statistics, and “sqlarbnd - Rebind” on page 23 to rebind the packages that use this table.

If the table is partitioned onto several nodes, and the table reorganization fails on any of the affected nodes, then only the failing nodes will have the table reorganization rolled back.

Note: If the reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.

If the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries.

REORGANIZE TABLE cannot be used on views.

REORGANIZE TABLE cannot be used on a DMS table while an online backup of a table space in which the table resides is being performed.

To complete a table space roll-forward recovery following a table reorganization, both data and LONG table spaces must be roll-forward enabled.

If the table contains LOB columns that do not use the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS/DMS).

DB2 Version 2 servers do not support down-level client requests to reorganize a table. Since pre-Version 2 servers do not support table spaces, the *pTablespace* parameter is treated as the Version 1 *path* parameter, when Version 2 clients are used with a down-level server.

If a Version 2 client requests to reorganize a table on a Version 2 server, and that request includes a path instead of a temporary table space in the *pTablespace* parameter (for example, an old application, specifying a temporary file path, being executed on Version 2 clients), REORG chooses a temporary table space in which to place the work files on behalf of the user. A valid temporary table space name containing a path separator character (/ or \) should not be specified, because it will be interpreted as a temporary path (pre-Version 2 request), and REORG will choose a temporary table space on behalf of the user.

See Also

"sqlarbnd - Rebind" on page 23
"sqlustat - Runstats" on page 319.

sqlurlog - Asynchronous Read Log

Provides the caller with the ability to extract certain log records from the DB2 Common Server database logs, and to query the Log Manager for current log state information. This API can only be used on databases with recoverable database logs (the configuration parameters LOGRETAIN or USEREXIT enabled).

Authorization

One of the following:

sysadm
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Asynchronous Read Log */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurlog (
    unsigned long CallerAction,
    SQLU_LSN * pStartLsn,
    SQLU_LSN * pEndLsn,
    char * pLogBuffer,
    unsigned long LogBufferSize,
    SQLU_RLOG_INFO * pReadLogInfo,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

CallerAction

Input. Specifies the action to be performed.

SQLU_RLOG_READ
Read the database log from the starting log sequence to the ending log sequence number and return all propagatable log records within this range.

SQLU_RLOG_READ_SINGLE
Read a single log record (propagatable or not) identified by the starting log sequence number.

SQLU_RLOG_QUERY
: Query the database log. Results of the query will be sent back via the SQLU_RLOG_INFO structure (see “SQLU-RLOG-INFO” on page 422).

pStartLsn
: 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.

pEndLsn
: Input. The ending log sequence number specifies the ending relative byte address for the reading of the log. This value must be greater than *startLsn*, and does not need to be the end of an actual log record.

pLogBuffer
: Output. The buffer where all the propagatable log records read within the specified range are stored sequentially. This buffer must be large enough to hold a single log record. As a guideline, this buffer should be a minimum of 32 bytes. Its maximum size is dependent on the size of the requested range. Each log record in the buffer is prefixed by a six byte lsn (log sequence number), representing the lsn of the following log record.

LogBufferSize
: Output. Specifies the size, in bytes, of the log buffer.

pReadLogInfo
: Output. A structure detailing information regarding the call and the database log. For more information about this structure, see “SQLU-RLOG-INFO” on page 422.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Usage Notes

If the requested action is to read the log, the caller will provide a Log Sequence Number (lsn) range and a buffer to hold the log records. The ASYNCHRONOUS READ LOG API reads the log sequentially, bounded by the requested lsn range, and returns log records associated with tables having the DATA CAPTURE option CHANGES, and an SQLU_RLOG_INFO structure with the current active log information. If the requested action is query, the API returns an SQLU_RLOG_INFO structure with the current active log information.

To use the Asynchronous Log Reader, first query the database log for a valid starting lsn. Following the query call, the read log information structure (SQLU-RLOG-INFO) will contain a valid starting lsn (in the initialLSN member), to be used on a read call. The end of the current active log will be in the curActiveLSN member of the read log information structure. The value used as the ending lsn on a read can be one of the following:

- The value of the curActiveLSN
- A value greater than initialLSN
- `FFFF FFFF FFFF` which is interpreted by the asynchronous log reader as the end of the current log.

For more information about the read log information structure, see “SQLU-RLOG-INFO” on page 422.

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 Common Server log records returned by **sqlurlog** can be found in Appendix F, “DB2 Common Server Log Records” on page 501.

After the initial read, in order to read the next sequential log record, add 1 to the last read lsn returned in SQLU-RLOG-INFO. Resubmit the call, with this new starting lsn and a valid ending lsn. The next block of records is then read. An sqlca code of SQLU_RLOG_READ_TO_CURRENT means the log reader has read to the end of the current active log.

sqluroll - Rollforward Database

Recovers a database by applying transactions recorded in the database log files. Called after a database or a table space backup has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either *logretain*, *userexit*, or both of these database configuration parameters must be set on) before the database can be recovered with roll-forward recovery.

Scope

In a multi-node environment, this API can only be called from the catalog node. A database or table space rollforward call specifying a point-in-time affects all nodes that are listed in the `db2nodes.cfg` file. A database or table space rollforward call specifying end of logs affects the nodes that are specified. If no nodes are specified, it affects all nodes that are listed in the `db2nodes.cfg` file.

Authorization

One of the following:

sysadm
sysctrl
sysmaint

Required Connection

None. This API establishes a database connection.

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Rollforward Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqluroll (
    struct rfwd_input * pRfwdInput,
    struct rfwd_output * pRfwdOuput,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Rollforward Database */
/* ... */
SQL_API_RC SQL_API_RN
  sqlgroll (
    struct grfwd_input * grfwdin,
    struct rfwd_output * rfwdout,
    struct sqlca * sqlca);

SQL_STRUCTURE grfwd_input
{
   unsigned short DbAliasLen,
   unsigned short StopTimeLen,
   unsigned short UserNameLen,
   unsigned short PasswordLen,
   unsigned short OverflowLogPathLen,
   unsigned long Version,
   char * pDbAlias,
   unsigned short CallerAction,
   char * pStopTime,
   char * pUserName,
   char * pPassword,
   char * pOverflowLogPath,
   unsigned short NumChngLgOvrflw,
   struct sqlurf_newlogpath * pChngLogOvrflw,
   unsigned short ConnectMode,
   struct sqlu_tablespace_bkrst_list * pTablespaceList,
   short AllNodeFlag,
   short NumNodes,
   SQL_PDB_NODE_TYPE * pNodeList,
   short NumNodeInfo
}
/* ... */
```

API Parameters

pRfwdInput
: Input. A pointer to the *rfwd_input* structure. For more information about this structure, see "RFWD-INPUT" on page 334.

pRfwdOutput
: Output. A pointer to the *rfwd_output* structure. For more information about this structure, see "RFWD-OUTPUT" on page 337.

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

StopTimeLen
: Input. A 2-byte unsigned integer representing the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

UserNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
: Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

OverflowLogPathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

Version
: Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

pDbAlias
: Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

CallerAction
: Input. Specifies action to be taken. Valid values (defined in `sqlutil`) are:

  SQLUM_ROLLFWD
  Rollforward to the point in time specified by *pPointInTime*. For database rollforward, the database is left in *rollforward-pending* state. For table space rollforward to a point in time, the table spaces are left in *rollforward-in-progress* state.

  SQLUM_STOP
  End roll-forward recovery. No new log records are processed and uncommitted transactions are backed out. The *rollforward-pending* state of the database or table spaces is turned off. Synonym is SQLUM_COMPLETE.

  SQLUM_ROLLFWD_STOP
  Rollforward to the point in time specified by *pPointInTime*, and end roll-forward recovery. The *rollforward-pending* state of the database or table spaces is turned off. Synonym is SQLUM_ROLLFWD_COMPLETE.

  SQLUM_QUERY
  Query values for *pNextArcFileName*, *pFirstDelArcFileName*, *pLastDelArcFileName*, and *pLastCommitTime*. Return database status and a node number.

  SQLUM_PARM_CHECK
  Validate parameters without performing the roll forward.

  SQLUM_CANCEL
  Cancel the rollforward operation that is currently running. The database or table space are put in recovery pending state.

  Note: This option cannot be used while the rollforward is actually running. It can be used if the rollforward is paused (that is, waiting for a STOP), or if a system failure occurred during the rollforward. It should be used with caution.

Rolling databases forward may require a load recovery using tape devices. The rollforward API will return with a warning message if user intervention on a device is required. The API can be called again with one of the following three caller actions:

SQLUM_LOADREC_CONTINUE
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

SQLUM_LOADREC_DEVICE_TERMINATE
Stop using the device that generated the warning message (for example, when there are no more tapes).

SQLUM_LOADREC_TERMINATE
Terminate all devices being used by load recovery.

pStopTime
Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for `SQLUM_QUERY`, `SQLUM_PARM_CHECK`, and any of the load recovery (`SQLUM_LOADREC_`*xxx*) caller actions.

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

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

pOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the *logpath* (see “sqlfxdb - Get Database Configuration” on page 201) before they can be used by this utility. This can be a problem if the user does not have sufficient space in the *logpath*. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the *logpath*, and then in the overflow log path. The log files needed for table space roll-forward recovery can be brought into either the *logpath* or the overflow log path. If the caller does not specify an overflow log path, the default value is the *logpath*. In a multi-node 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-node environment, the overflow log path can be relative if the server is local.

NumChngLgOvrflw
MPP only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified node only.

pChngLogOvrflw
MPP 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 node only.

ConnectMode
Input. Valid values (defined in `sqlutil`) are:

SQLUM_OFFLINE
Offline roll forward. This value must be specified for database roll-forward recovery.

SQLUM_ONLINE
Online roll forward.

pTablespaceList
Input. A pointer to a structure containing the names of the table spaces to be rolled forward to the end-of-logs or to a specific point in time. If not specified, the table spaces needing rollforward will be selected.

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

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

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

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

SQLURF_CAT_NODE_ONLY
Apply to the catalog node only. *pNodeList* should be NULL.

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

pNodeList
Input. A pointer to an array of node numbers on which to perform the roll-forward recovery.

NumNodeInfo
Input. Defines the size of the output parameter *pNodeInfo*, which must be large enough to hold status information from each node that is being rolled forward. In a single-node environment, this parameter should be set to 1.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
ROLLFORWARD DATABASE database-alias [USING :value] [USER username USING password]
[rollforward_action_clause | load_recovery_action_clause]

where rollforward_action_clause stands for:

    { TO point-in-time [AND STOP] |
      {
        [TO END OF LOGS [AND STOP] | STOP | CANCEL | QUERY STATUS | PARM CHECK }
        [ON {:nodelist | ALL NODES [EXCEPT :nodelist]}]
      }
    }
    [TABLESPACE {ONLINE |:tablespacenames [ONLINE]} ]
    [OVERFLOW LOG PATH default-log-path [:logpaths]]

and load_recovery_action_clause stands for:

    LOAD RECOVERY { CONTINUE | DEVICE_TERMINATE | TERMINATE }
```

REXX API Parameters

database-alias
Alias of the database to be rolled forward.

value
A compound REXX host variable containing the output values. In the following, XXX represents the host variable name:

XXX.0	Number of elements in the variable
XXX.1	The application ID
XXX.2	Number of replies received from nodes
XXX.2.1.1	First node number
XXX.2.1.2	First state information
XXX.2.1.3	First next archive file needed
XXX.2.1.4	First first archive file to be deleted
XXX.2.1.5	First last archive file to be deleted
XXX.2.1.6	First last commit time
XXX.2.2.1	Second node number
XXX.2.2.2	Second state information
XXX.2.2.3	Second next archive file needed
XXX.2.2.4	Second first archive file to be deleted
XXX.2.2.5	Second last archive file to be deleted
XXX.2.2.6	Second last commit time

XXX.2.3.x	and so on.

username
Identifies the user name under which the database is to be rolled forward.

password
The password used to authenticate the user name.

point-in-time
A time stamp in ISO format, *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes,seconds, microseconds), expressed in Coordinated Universal Time (UTC).

tablespacenames
A compound REXX host variable containing a list of table spaces to be rolled forward. In the following, XXX is the name of the host variable:

XXX.0	Number of table spaces to be rolled forward
XXX.1	First table space name
XXX.2	Second table space name
XXX.x	and so on.

default-log-path
The default overflow log path to be searched for archived logs during recovery

logpaths
A compound REXX host variable containing a list of alternate log paths to be searched for archived logs during recovery. In the following, XXX is the name of the host variable:

XXX.0	Number of changed overflow log paths
XXX.1.1	First node
XXX.1.2	First overflow log path
XXX.2.1	Second node
XXX.2.2	Second overflow log path
XXX.3.1	Third node
XXX.3.2	Third overflow log path
XXX.x.1	and so on.

nodelist
A compound REXX host variable containing a list of nodes. In the following, XXX is the name of the host variable:

XXX.0	Number of nodes
XXX.1	First node
XXX.2	Second node
XXX.x	and so on.

Sample Programs

C	\sqllib\samples\c\backrest.c
COBOL	\sqllib\samples\cobol\backrest.cbl
FORTRAN	\sqllib\samples\fortran\backrest.f

Usage Notes

The database manager uses the information stored in the archived and the active log files to reconstruct the transactions performed on the database since its last backup.

The action performed when this API is called depends on the *rollforward_pending* flag of the database prior to the call. This can be queried using "sqlfxdb - Get Database Configuration" on page 201. The *rollforward_pending* flag is set to `DATABASE` if the database is in roll-forward pending state. It is set to `TABLESPACE` if one or more table spaces are in `SQLB_ROLLFORWARD_PENDING` or `SQLB_ROLLFORWARD_IN_PROGRESS` state. The *rollforward_pending* flag is set to `NO` if neither the database nor any of the table spaces needs to be rolled forward.

If the database is in roll-forward pending state when this API is called, the database will be rolled forward. Table spaces are returned to normal state after a successful database roll-forward, unless an abnormal state causes one or more table spaces to go offline. If the *rollforward_pending* flag is set to `TABLESPACE`, only those table spaces that are in roll-forward pending state, or those table spaces requested by name, will be rolled forward.

Note: If table space rollforward terminates abnormally, table spaces that were being rolled forward will be put in `SQLB_ROLLFORWARD_IN_PROGRESS` state. In the next invocation of ROLLFORWARD DATABASE, only those table spaces in `SQLB_ROLLFORWARD_IN_PROGRESS` state will be processed. If the set of selected table space names does not include all table spaces that are in `SQLB_ROLLFORWARD_IN_PROGRESS` state, the table spaces that are not required will be put into `SQLB_RESTORE_PENDING` state.

If the database is not in roll-forward pending state and no point in time is specified, any table spaces that are in rollforward-in-progress state will be rolled forward to the end of logs. If no table spaces are in rollforward-in-progress state, any table spaces that are in rollforward pending state will be rolled forward to the end of logs.

This API reads the log files, beginning with the log file that is matched with the backup image. The name of this log file can be determined by calling this API with a caller action of `SQLUM_QUERY` before rolling forward any log files.

The transactions contained in the log files are reapplied to the database. The log is processed as far forward in time as information is available, or until the time specified by the stop time parameter.

Recovery stops when any one of the following events occurs:

- No more log files are found

- A time stamp in the log file exceeds the completion time stamp specified by the stop time parameter
- An error occurs while reading the log file.

Some transactions might not be recovered. The value returned in *pLastCommitTime* indicates the time stamp of the last committed transaction that was applied to the database.

If the need for database recovery was caused by application or human error, the user may want to provide a time stamp value in *pStopTime*, indicating that recovery should be stopped before the time of the error. This applies only to full database roll-forward recovery, and to table space rollforward to a point in time. It also permits recovery to be stopped before a log read error occurs, determined during an earlier failed attempt to recover.

When the *rollforward_recovery* flag is set to `DATABASE`, the database is not available for use until roll-forward recovery is terminated. Termination is accomplished by calling the API with a caller action of `SQLUM_STOP` or `SQLUM_ROLLFORWARD_STOP` to bring the database out of roll-forward pending state. If the *rollforward_recovery* flag is `TABLESPACE`, the database is available for use. However, the table spaces in `SQLB_ROLLFORWARD_PENDING` and `SQLB_ROLLFORWARD_IN_PROGRESS` states will not be available until the API is called to perform table space roll-forward recovery. If rolling forward table spaces to a point in time, the table spaces are placed in backup pending state after a successfull rollforward.

Rolling databases forward may involve prerequisites and restrictions that are beyond the scope of this manual. For more detailed information, see the *Administration Guide*.

See Also

"sqluload - Load" on page 282
"sqlurst - Restore Database" on page 309.

sqlurst - Restore Database

Rebuilds a damaged or corrupted database that has been backed up using BACKUP DATABASE. The restored database is in the same state it was in when the backup copy was made. This utility can also restore to a database with a name different from the database name in the backup image (in addition to being able to restore to a new database).

The utility can also be used to restore previous versions of DB2 databases.

If, at the time of the backup operation, the database was enabled for roll-forward recovery, the database can be brought to the state it was in prior to the occurrence of the damage or corruption by issuing **sqluroll** after successful execution of **sqlurst**.

This utility can also restore from a table space level backup.

Scope

This API only affects the node from which it is called.

Authorization

To restore to an existing database, one of the following:

sysadm
sysctrl
sysmaint

To restore to a new database, one of the following:

sysadm
sysctrl

Required Connection

Database, to restore to an existing database. This API automatically establishes a connection to the specified database.

Instance and database, to restore to a new database. The instance attachment is required to create the database.

To restore to a new database at an instance different from the current instance (as defined by the value of the **DB2INSTANCE** environment variable), it is necessary to first attach to the instance where the new database will reside.

API Include File

sqlutil.h

sqlurst - Restore Database

C API Syntax

```
/* File: sqlutil.h */
/* API: Restore Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlurst (
    char * pSourceDbAlias,
    char * pTargetDbAlias,
    unsigned long BufferSize,
    unsigned long RollforwardMode,
    unsigned long RestoreType,
    unsigned long RestoreMode,
    unsigned long CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    char * pTargetPath,
    unsigned long NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaSourceList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    unsigned long VendorOptionsSize,
    void * pVendorOptions,
    unsigned long Parallelism,
    void * pRestoreInfo,
    void * pContainerPageList,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Restore Database */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgrst (
    unsigned short SourceDbAliasLen,
    unsigned short TargetDbAliasLen,
    unsigned short TimestampLen,
    unsigned short TargetPathLen,
    unsigned short UserNameLen,
    unsigned short PasswordLen,
    unsigned short * pReserved1,
    char * pSourceDbAlias,
    char * pTargetDbAlias,
    unsigned long BufferSize,
    unsigned long RollforwardMode,
    unsigned long RestoreType,
    unsigned long RestoreMode,
    unsigned long CallerAction,
    char * pApplicationId,
    char * pTimestamp,
    char * pTargetPath,
    unsigned long NumBuffers,
    struct sqlu_tablespace_bkrst_list * pTablespaceList,
    struct sqlu_media_list * pMediaSourceList,
    char * pUserName,
    char * pPassword,
    void * pReserved2,
    unsigned long Parallelism,
    unsigned short RestoreInfoSize,
    void * pRestoreInfo,
    unsigned short ContainerPageListSize,
    void * pContainerPageList,
    unsigned long VendorOptionsSize,
    void * pVendorOptions,
    void * pReserved3,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

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

TargetDbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the target database alias. Set to zero if no target database alias is specified.

TimestampLen
: Input. A 2-byte unsigned integer representing the length in bytes of the time stamp. Set to zero if no time stamp is provided.

TargetPathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the target directory. Set to zero if no target path is provided.

UserNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

PasswordLen
: Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

pReserved1
: Reserved for future use.

pSourceDbAlias
: Input. A string containing the database alias of the source database backup image.

pTargetDbAlias
: Input. A string containing the target database alias. If this parameter is null, the *pSourceDbAlias* alias is used.

BufferSize
: Input. Restore buffer size in allocation units of 4KB. Minimum is 16 units.

RollforwardMode
: Input. Indicates whether or not to place the database in rollforward pending state at the end of the restore. Valid values (defined in `sqlutil`) are:

  SQLUD_ROLLFWD
  Place the database in roll-forward pending state after it has been successfully restored.

  SQLUD_NOROLLFWD
  Do not place the database in roll-forward pending state after it has been successfully restored.

  If, following a successful restore, the database is in roll-forward pending state, “sqluroll - Rollforward Database” on page 300 must be executed before the database can be used.

RestoreType
: Input. Specifies the type of restore. Valid values (defined in `sqlutil`) are:

  SQLUD_FULL
  Restore everything from the backup image. This will be run offline.

  SQLUD_ONLINE_TABLESPACE
  Restore only the table space level backups. This will be run online.

  SQLUD_HISTORY
  Restore only the recovery history file.

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

  SQLUD_RESTORE
  Start the restore.

SQLUD_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, when all of the media required for the restore are known to have been mounted, and utility prompts are not desired.

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

SQLUD_TERMINATE
Terminate the restore after the user has failed to perform some action requested by the utility.

SQLUD_DEVICE_TERMINATE
Remove a particular device from the list of devices used by the restore utility. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action, and the device which generated the warning will be removed from the list of devices being used.

SQLUD_PARM_CHECK
Validate parameters without performing the restore.

SQLUD_RESTORE_STORDEF
Initial call. Table space container redefinition requested.

CallerAction must be set to `SQLUD_RESTORE`, `SQLUD_NOINTERRUPT`, `SQLUD_RESTORE_STORDEF`, or `SQLUD_PARM_CHECK` on the first call.

pApplicationId
Output. Supply a buffer of length `SQLU_APPLID_LEN+1` (defined in `sqlutil`). Restore will return a string identifying the agent servicing the application. Can be used with the database system monitor APIs to monitor some aspects of the application.

pTimestamp
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.

pTargetPath
Input. A string containing the relative or fully qualified name of the target database directory. Used if a new database is to be created for the restored backup.

NumBuffers
Input. The number of buffers to be used for the restore.

pTablespaceList
Specifies one or more table spaces to be restored. Used when restoring a subset of the backup image.

The following restrictions apply:
- The backup image must have been created by DB2 Version 3.
- The database must be recoverable; that is, log retain or user exits must be enabled.

- The database being restored to must be the same database that was used to create the backup image.
- This function is not supported by back level APIs.
- This function is not available when restoring from a user exit on OS/2.
- The rollforward utility will ensure that table spaces restored in an MPP environment are synchronized with any other node containing the same table spaces.

pMediaSourceList
Input. Source media for the backup image. See structure "SQLU-MEDIA-LIST" on page 417. The information the caller needs to provide in this structure is dependent upon the value of the *media_type* field. Valid values for this field (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
Local devices (a combination of tapes, disks or diskettes). Provide a list of *sqlu_media_entry*. On OS/2 or the Windows operating system, the entries can be directory paths only, not tape device names.

SQLU_ADSM_MEDIA
ADSM. No additional input is required, and the ADSM shared library provided with DB2 is used. If a different version of ADSM is desired, use `SQLU_OTHER_MEDIA` and provide the shared library name.

SQLU_OTHER_MEDIA
Vendor product. Provide the shared library name in an *sqlu_vendor* structure.

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

For more information, see the *Administration Guide*.

pUserName
Input. A string containing the user name to be used for connection.

pPassword
Input. A string containing the password to be used with the user name for connection.

pReserved2
Reserved for future use.

Parallelism
Input. Degree of parallelism (number of buffer manipulators).

RestoreInfoSize
Reserved for future use.

pRestoreInfo
Reserved for future use.

ContainerPageListSize
Reserved for future use.

pContainerPageList
Reserved for future use.

VendorOptionsSize
Input. The length of the options field.

pVendorOptions
Input. To be used by the vendor 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 for this data is not checked.

pReserved3
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

REXX API Syntax

```
RESTORE DATABASE source-database-alias [USING :value] [USER username USING password]

[TABLESPACE :tablespacenames] [ONLINE | HISTORY FILE ]

[LOAD shared-library [OPTIONS vendor-options] [OPEN num-sessions SESSIONS] |
 FROM :source-area | USE ADSM [OPEN num-sessions SESSIONS] | USER_EXIT]

[TAKEN AT timestamp] [TO target-directory] [INTO target-database-alias]

[ACTION caller-action] [WITH num-buffers BUFFERS] [BUFFERSIZE buffer-size]

[WITHOUT ROLLING FORWARD] [PARALLELISM parallelism-degree]
```

REXX API Parameters

source-database-alias
: Alias of the source database from which the database backup image was taken.

value
: A compound REXX host variable to which the database restore information is returned. In the following, XXX represents the host variable name:

XXX.0	Number of elements in the variable (always 1)
XXX.1	An application ID that identifies the agent that serves the application.

username
: Identifies the user name to be used for connection.

password
: The password used to authenticate the user name.

tablespacenames
: A compound REXX host variable containing a list of table spaces to be restored. In the following, XXX is the name of the host variable:

XXX.0	Number of table spaces to be restored
XXX.1	First table space name
XXX.2	Second table space name
XXX.3	and so on.

HISTORY FILE
: Specifies to restore the history file from the backup.

shared-library
: The name of the shared library (DLL on OS/2 or the Windows operating system) containing the vendor restore I/O functions to be used. It may contain the full path. If the full path is not given, defaults to the path on which the user exit program resides.

vendor-options
: Information required by the vendor functions.

num-sessions
: The number of I/O sessions to be used with ADSM or the vendor product.

source-area
: A compound REXX host variable that indicates on which directory or device the backup image resides. The default value is the current directory. On OS/2 or the Windows operating system, the entries can be directory paths only, not tape device names.

timestamp
: The time stamp of the database backup.

target-directory
: The directory of the target database.

target-database-alias
: Alias of the target database. If the target database does not exist, it will be created.

caller-action
: Specifies action to be taken. Valid values are:

SQLUD_RESTORE
Start the restore.

SQLUD_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, when all of the media required for the restore are known to have been mounted, and utility prompts are not desired.

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

SQLUD_TERMINATE
Terminate the restore after the user has failed to perform some action requested by the utility.

SQLUD_DEVICE_TERMINATE
Remove a particular device from the list of devices used by the restore utility. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action, and the device which generated the warning will be removed from the list of devices being used.

SQLUD_PARM_CHECK
Validate parameters without performing the restore.

SQLUD_RESTORE_STORDEF
Initial call. Table space container redefinition requested.

num-buffers
Number of backup buffers to be used.

buffer-size
Backup buffer size in allocation units of 4KB. Minimum is 16 units.

parallelism-degree
Number of buffer manipulators.

Sample Programs

C	\sqllib\samples\c\backrest.c
COBOL	\sqllib\samples\cobol\backrest.cbl
FORTRAN	\sqllib\samples\fortran\backrest.f

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 operating system restore utility is being used to perform the restore, and any application, including the calling application, is already connected to any database on the same workstation. If the connect is successful, the API locks out other applications until the restore is completed.

The current database configuration file will not be replaced by the backup copy unless it is unusable. If the file is replaced, a warning message is returned.

The database or table space must have been backed up using "sqlubkp - Backup Database" on page 230.

If the caller action is `SQLUD_NOINTERRUPT`, the restore continues without prompting the application. If the caller action is `SQLUD_RESTORE`, and the utility is restoring to an existing database, the utility returns control to the application with a message requesting some user interaction. After handling the user interaction, the application calls RESTORE DATABASE again, with the caller action set to indicate whether processing is to continue (`SQLUD_CONTINUE`) or terminate (`SQLUD_TERMINATE`) on the subsequent call. The utility finishes processing, and returns an SQLCODE in the *sqlca*.

To close a device when finished, set the caller action to `SQLUD_DEVICE_TERMINATE`. If, for example, a user is restoring from 3 tape volumes using 2 tape devices, and one of the tapes has been restored, the application obtains control from the API with an SQLCODE indicating end of tape. The application can prompt the user to mount another tape, and if the user indicates "no more", return to the API with caller action `SQLUD_DEVICE_TERMINATE` to signal end of the media device. The device driver will be terminated, but the rest of the devices involved in the restore will continue to have their input processed until all segments of the restore set have been restored (the number of segments in the restore set is placed on the last media device during the backup process). This caller action can be used with devices other than tape (vendor supported devices).

To perform a parameter check before returning to the application, set caller action to `SQLUD_PARM_CHECK`.

Set caller action to `SQLUD_RESTORE_STORDEF` when performing a redirected restore; used in conjunction with “sqlbstsc - Set Tablespace Containers” on page 51. For more information, see the *Administration Guide*.

If an error occurs, the utility terminates and returns the error in the *sqlca* structure.

If a system failure occurs during a critical stage of restoring a database, the user will not be able to successfully connect to the database until a successful restore is performed. This condition will be detected when the connection is attempted, and an error message is returned. If the backed-up database is not configured for roll-forward recovery, and there is a usable current configuration file with either of these parameters enabled, following the restore, the user will be required to either take a new backup of the database, or disable the log retain and user exit parameters before connecting to the database.

Although the restored database will not be dropped (unless restoring to a non-existent database), if the restore fails, it will not be usable.

If the restore type specifies that the recovery history file on the backup is to be restored, it will be restored over the existing recovery 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.

See Also

“sqlbstsc - Set Tablespace Containers” on page 51
“sqlemgdb - Migrate Database” on page 145
“sqlfxdb - Get Database Configuration” on page 201
“sqlubkp - Backup Database” on page 230
“sqluroll - Rollforward Database” on page 300.

sqlustat - Runstats

Updates statistics about the characteristics of a table and any associated indexes. These characteristics include, among many others, number of records, number of pages, and average record length. The optimizer uses these statistics when determining access paths to the data.

This utility should be called when a table has had many updates, after reorganizing a table, or after creating a new index.

Statistics are collected based on the table partition that is resident on the node where the API executes. Global table statistics are derived by multiplying the values obtained at a node by the number of nodes on which the table is completely stored. The global statistics are stored in the catalog tables.

The node from which the API is called does not have to contain a partition for the table:

- If the API is called from a node that contains a partition for the table, the utility executes at this node.
- If the API is called from a node that does not contain a table partition, the request is sent to the first node in the nodegroup that holds a partition for the table. The utility then executes at this node.

Scope

This API can be called from any node in the `db2nodes.cfg` file. It can be used to update the catalogs on the catalog node.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm
CONTROL privilege on the table.

Required Connection

Database

API Include File

sqlutil.h

sqlustat - Runstats

C API Syntax

```
/* File: sqlutil.h */
/* API: Run Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlustat (
    _SQLOLDCHAR * pTableName,
    unsigned short NumIndexes,
    _SQLOLDCHAR ** ppIndexList,
    unsigned char StatsOption,
    unsigned char ShareLevel,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Run Statistics */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgstat (
    unsigned short TableNameLen,
    unsigned short NumIndexes,
    unsigned char StatsOption,
    unsigned char ShareLevel,
    unsigned short * pIndexLens,
    struct sqlca * pSqlca,
    _SQLOLDCHAR ** ppIndexList,
    _SQLOLDCHAR * pTableName);
/* ... */
```

API Parameters

TableNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the table name.

NumIndexes
: Input. The number of indexes specified in this call. This value is used with the *StatsOption* parameter. Valid values are:

0
: All the indexes are to be calculated.

n
: The number of indexes contained in the index list. The names of the indexes to be calculated are specified in *ppIndexList*.

StatsOption
: Input. Statistical option, indicating which calculations are to be performed. Valid values (defined in `sqlutil`) are:

  SQL_STATS_TABLE
  : Table only.

  SQL_STATS_EXTTABLE_ONLY
  : Table with extended (distribution) statistics.

  SQL_STATS_BOTH
  : Both table and indexes.

  SQL_STATS_EXTTTABLE_INDEX
  : Both table (with distribution statistics) and basic indexes.

  SQL_STATS_INDEX
  : Indexes only.

  SQL_STATS_EXTINDEX_ONLY
  : Extended statistics for indexes only.

  SQL_STATS_EXTINDEX_TABLE
  : Extended statistics for indexes and basic table statistics.

  SQL_STATS_ALL
  : Extended statistics for indexes and table statistics with distribution statistics.

ShareLevel
: Input. Specifies how the statistics are to be gathered with respect to other users. Valid values (defined in `sqlutil`) are:

  SQL_STATS_REF
  : Allows others to have read-only access while the statistics are being gathered.

  SQL_STATS_CHG
  : Allows others to have read and write access while the statistics are being gathered.

pIndexLens
: Input. An array of 2-byte unsigned integers representing the length in bytes of each of the index names in the index list.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

ppIndexList
: Input. An array of strings. Each string contains one fully qualified index name.

pTableName
: Input. The table on which to update statistics. Can be an alias, except in the case of down-level servers, when the fully qualified table name must be used.

sqlustat - Runstats

REXX API Syntax

```
RUNSTATS ON TABLE tname
[WITH :statsopt INDEXES {ALL | USING :value}]
[SHRLEVEL {REFERENCE|CHANGE}]
```

REXX API Parameters

tname
: The fully qualified name of the table on which statistics are to be gathered.

statsopt
: A host variable containing a statistical option, indicating which calculations are to be performed. Valid values are:

T	Indicates that basic statistics are to be updated for the specified table only. This is the default
D	Indicates that extended (distribution) statistics are to be updated for the specified table
B	Indicates that basic statistics are to be updated for both the specified table and the specified indexes
E	Indicates that extended statistics are to be updated for the specified table, and that basic statistics are to be updated for the indexes
I	Indicates that basic statistics are to be updated for the specified indexes only
X	Indicates that extended statistics are to be updated for the specified indexes only
Y	Indicates that basic statistics are to be updated for the specified table, and that extended statistics are to be updated for the indexes
A	Indicates that extended statistics are to be updated for both the specified table and the specified indexes.

value
: A compound REXX host variable containing the names of the indexes for which statistics are to be generated. In the following, XXX represents the host variable name:

XXX.0	The number of indexes specified in this call
XXX.1	First fully qualified index name
XXX.2	Second fully qualified index name
XXX.3	and so on.

REFERENCE
: Other users can have read-only access while updates are being made.

CHANGE
: Other users can have read or write access while updates are being made. This is the default.

Sample Programs

C	\sqllib\samples\c\dbstat.sqc
COBOL	\sqllib\samples\cobol\dbstat.sqb
FORTRAN	\sqllib\samples\fortran\dbstat.sqf

Usage Notes

Use RUNSTATS to update statistics:

- On tables that have been modified many times (for example, if a large number of updates have been made, or if a significant amount of data has been inserted or deleted)
- On tables that have been reorganized
- When a new index has been created.

After statistics have been updated, new access paths to the table can be created by rebinding the packages using “sqlabndx - Bind” on page 10.

Statistics for tables only should be collected before any indexes are created. This will ensure that statistics gathered during index creation are not overlaid by estimates gathered during the calculation of table statistics.

If index statistics are requested, and statistics have never been run on the table containing the index, statistics on both the table and indexes are calculated.

After calling this API the application should issue a COMMIT to release the locks.

To allow new access plans to be generated, the packages that reference the target table must be rebound after using this API.

In FORTRAN, use “sqlgaddr - Get Address” on page 207 to initialize the pointers in the index list.

See Also

"REORGCHK" in the *Command Reference*
“sqlfxdb - Get Database Configuration” on page 201
“sqlureot - Reorganize Table” on page 293.

sqluvqdp - Quiesce Tablespaces for Table

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive. There are three possible table space states resulting from the quiesce function: QUIESCED SHARE, QUIESCED UPDATE, and QUIESCED EXCLUSIVE.

Scope

In a single-node environment, this API quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load. In an MPP environment, this API acts locally on a node. It quiesces only that portion of table spaces belonging to the node on which the load is performed.

Authorization

One of the following:

sysadm
sysctrl
sysmaint
dbadm

Required Connection

Database

API Include File

sqlutil.h

C API Syntax

```
/* File: sqlutil.h */
/* API: Quiesce Tablespaces for Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqluvqdp (
    char * pTableName,
    long QuiesceMode,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Quiesce Tablespaces for Table */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgvqdp (
    unsigned short TableNameLen,
    char * pTableName,
    long QuiesceMode,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

TableNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the table name.

pTableName
Input. A string containing the table name as used in the system catalog. This may be a two-part name with the *schema* and the table name separated by a period (.). If the *schema* is not provided, the authorization ID used in the connection will be used as the *schema*. The table cannot be a system catalog table. This field is mandatory.

QuiesceMode
Input. Specifies the quiesce mode. Valid values (defined in `sqlutil`) are:

SQLU_QUIESCEMODE_SHARE
For share mode

SQLU_QUIESCEMODE_INTENT_UPDATE
For intent to update mode

SQLU_QUIESCEMODE_EXCLUSIVE
For exclusive mode

SQLU_QUIESCEMODE_RESET
To reset the state of the table spaces to normal if either of the following is true:
- The caller owns the quiesce
- The caller who sets the quiesce disconnects, creating a "phantom quiesce"

SQLU_QUIESCEMODE_RESET_OWNED
To reset the state of the table spaces to normal if the caller owns the quiesce.

This field is mandatory.

pReserved
Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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 authorization ID used in the connection will be used as the *schema*.

Sample Programs

C	\sqllib\samples\c\tload.sqc
COBOL	\sqllib\samples\cobol\tload.sqb
FORTRAN	\sqllib\samples\fortran\tload.sqf
REXX	\sqllib\samples\rexx\quitab.cmd

Usage Notes

When the quiesce share request is received, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces is recorded in the table space table, along with the authorization ID and the database agent ID of the quiescer, so that the state is persistent.

The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces will be allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

When the quiesce exclusive request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer), however, has exclusive access to the table and the table spaces.

When a quiesce update request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces with the quiescer is recorded in the table space table.

There is a limit of five quiescers on a table space at any given time. Since QUIESCED EXCLUSIVE is incompatible with any other state, and QUIESCED UPDATE is incompatible with another QUIESCED UPDATE, the five quiescer limit, if reached, must have at least four QUIESCED SHARE and at most one QUIESCED UPDATE.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

The quiesced state of a table space must be reset explicitly by using SQLU_QUIESCEMODE_RESET.

See Also

"sqluload - Load" on page 282.

Chapter 2. Additional REXX APIs

This chapter describes DB2 application programming interfaces that are only supported in the REXX programming language.

Change Isolation Level

Changes the way that DB2 isolates data from other processes while a database is being accessed.

Authorization

None

Required Connection

None

REXX API Syntax

```
CHANGE SQLISL TO {RR|CS|UR|RS|NC}
```

REXX API Parameters

RR
: Repeatable read.

CS
: Cursor stability. This is the default.

UR
: Uncommitted read.

RS
: Read stability.

NC
: No commit.

Sample Program

REXX	\sqllib\samples\rexx\chgisl.cmd

Chapter 3. Data Structures

This chapter describes the data structures used to access the database manager. The following data structures are provided:

“SQLE-NODE-NETB” on page 375
: Passes information for cataloging NetBIOS nodes

“SQLE-NODE-NPIPE” on page 376
: Passes information for cataloging named pipe nodes

“SQLE-NODE-STRUCT” on page 377
: Passes information for cataloging nodes

“SQLE-NODE-TCPIP” on page 379
: Passes information for cataloging TCP/IP nodes

“SQLE-REG-NWBINDERY” on page 380
: Passes information for registering/deregistering the DB2 server in/from the bindery on the NetWare file server

“SQLE-START-OPTIONS” on page 381
: Holds the database manager start-up options

“SQLEDBCOUNTRYINFO” on page 385
: Transfers country information between an application and the database manager

“SQLEDBDESC” on page 386
: Passes creation parameters to the CREATE DATABASE API

“SQLEDBSTOPOPT” on page 392
: Holds the database manager stop options

“SQLEDINFO” on page 394
: Returns a copy of a single directory entry from the system or local database directory

“SQLENINFO” on page 397
: Returns a copy of a single directory entry from the node directory

“SQLFUPD” on page 400
: Passes configuration file information

“SQLM-COLLECTED” on page 407
: Transfers Database System Monitor collection count information between an application and the database manager

“SQLM-RECORDING-GROUP” on page 410
: Transfers Database System Monitor monitor group information between an application and the database manager

“SQLMA” on page 412
: Sends database monitor requests from an application to the database manager

“SQLOPT” on page 414
: Transfers bind parameters to the BIND API and precompile options to the PRECOMPILE PROGRAM API

“SQLU-LSN” on page 416
: Contains the definition of the log sequence number used by the ASYNCHRONOUS READ LOG API

“SQLU-MEDIA-LIST” on page 417
: Holds a list of target media (BACKUP) or source media (RESTORE) for the backup image. Also used for the import, export and load APIs

“SQLU-RLOG-INFO” on page 422
: Contains information regarding a call to the ASYNCHRONOUS READ LOG API

“SQLU-TABLESPACE-BKRST-LIST” on page 423
: Provides a list of table space names

“SQLUEXPT-OUT” on page 425
: Transfers export information between an application and the database manager

“SQLUHINFO” on page 426
: Used by the recovery history file APIs to return information from the recovery history file

“SQLUIMPT-IN” on page 430
: Transfers import information between an application and the database manager

“SQLUIMPT-OUT” on page 431
: Transfers import information between an application and the database manager

“SQLULOAD-IN” on page 433
: Transfers load information between an application and the database manager

“SQLULOAD-OUT” on page 437
: Transfers load information between an application and the database manager

“SQLUPI” on page 439
: Contains partitioning information, such as the partitioning map and the partitioning key of a table

“SQLXA-RECOVER” on page 441
: Used by the transaction APIs to return a list of indoubt transactions

“SQLXA-XID” on page 443
: Used by the transaction APIs to identify a transaction.

RFWD-INPUT

This structure is used to pass information to “sqluroll - Rollforward Database” on page 300.

Table 6. Fields in the RFWD-INPUT Structure

Field Name	**Data Type**	**Description**
VERSION	UNSIGNED LONG	Rollforward version.
PDBALIAS	Pointer	Database alias.
CALLERACTION	UNSIGNED SHORT	Action.
PSTOPTIME	Pointer	Stop time.
PUSERNAME	Pointer	User name.
PPASSWORD	Pointer	Password.
POVERFLOWLOGPATH	Pointer	Overflow log path.
NUMCHNGLGOVRFLW	UNSIGNED SHORT	Number of changed overflow log paths (MPP only).
PCHNGLOGOVRFLW	Structure	Changed overflow log paths (MPP only).
CONNECTMODE	UNSIGNED SHORT	Connect mode.
PTABLESPACELIST	Structure	A pointer to a list of table space names. For information about this structure, see “SQLU-TABLESPACE-BKRST-LIST” on page 423.
ALLNODEFLAG	SHORT	All node flag.
NUMNODES	SHORT	Size of the node list.
PNODELIST	Pointer	List of node numbers.
NUMNODEINFO	SHORT	Size of *pNodeInfo* in “RFWD-OUTPUT” on page 337.
NODENUM	SQL_PDB_NODE_TYPE	Node number.
PATHLEN	UNSIGNED SHORT	Length of the new log path.
LOGPATH	CHAR(255)	New overflow log path.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: RFWD-INPUT */
/* ... */
SQL_STRUCTURE rfwd_input
{
  unsigned long   version;
  char            *pDbAlias;
  unsigned short  CallerAction;
  char            *pStopTime;
  char            *pUserName;
  char            *pPassword;
  char            *pOverflowLogPath;
  unsigned short  NumChngLgOvrflw;
  struct sqlurf_newlogpath *pChngLogOvrflw;
  unsigned short  ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  short           AllNodeFlag;
  short           NumNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  short           NumNodeInfo;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLURF-NEWLOGPATH */
/* ... */
SQL_STRUCTURE sqlurf_newlogpath
{
  SQL_PDB_NODE_TYPE nodenum;
  unsigned short  pathlen;
  char            logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-RFWD-INPUT.
    05 SQL-VERSION           PIC 9(9) COMP-5.
    05 SQL-DBALIAS           USAGE IS POINTER.
    05 SQL-CALLERACTION      PIC 9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-STOPTIME          USAGE IS POINTER.
    05 SQL-USERNAME          USAGE IS POINTER.
    05 SQL-PASSWORD          USAGE IS POINTER.
    05 SQL-OVERFLOWLOGPATH   USAGE IS POINTER.
    05 SQL-NUMCHANGE         PIC 9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-P-CHNG-LOG-OVRFLW USAGE IS POINTER.
    05 SQL-CONNECTMODE       PIC 9(4) COMP-5.
    05 FILLER                PIC X(2).
    05 SQL-P-TABLESPACE-LIST USAGE IS POINTER.
    05 SQL-ALLNODEFLAG       PIC S9(4) COMP-5.
    05 SQL-NUMNODES          PIC S9(4) COMP-5.
    05 SQL-NODELIST          USAGE IS POINTER.
    05 SQL-NUMNODEINFO       PIC S9(4) COMP-5.
    05 FILLER                PIC X(2).
*
```

```
* File: sqlutil.cbl
01 SQLURF-NEWLOGPATH.
    05 SQL-NODENUM           PIC S9(4) COMP-5.
    05 SQL-PATHLEN           PIC 9(4) COMP-5.
    05 SQL-LOGPATH           PIC X(254).
    05 FILLER                PIC X.
    05 FILLER                PIC X(1).
*
```

RFWD-OUTPUT

This structure is used to pass information from "sqluroll - Rollforward Database" on page 300.

Table 7 (Page 1 of 2). Fields in the RFWD-OUTPUT Structure

Field Name	**Data Type**	**Description**
PAPPLICATIONID	Pointer	The address of a buffer of length `SQLU_APPLID_LEN+1` (defined in `sqlutil`) to hold an application identifier returned from the API. This identifier can be used with the database system monitor APIs to monitor some aspects of the application. If this information is not of interest, supply the NULL pointer. In a multi-node environment, returns only the application identifier for the catalog node.
PNUMREPLIES	Pointer	Number of node replies received. Each node that replies fills in an *sqlurf_info* structure in *pNodeInfo*. In a single-node environment, the value of this parameter is 1.
PNODEINFO	Structure	Node reply information. A user defined array of *NumNodeInfo sqlurf_info* structures.
NODENUM	SQL_PDB_NODE_TYPE	Node number.
STATE	LONG	State information.
NEXTARCLOG	UNSIGNED CHAR(13)	A 12-byte buffer to hold the returned name of the next archived log file required. If a caller action other than `SQLUM_QUERY` is supplied, the value returned in this field indicates that an error occurred when accessing the file. Possible causes are: • The file was not found in the database log directory, nor on the path specified by the overflow log path parameter • The user exit program failed to return the archived file.

FIRSTARCDEL	UNSIGNED CHAR(13)	A 12-byte buffer to hold the returned name of the first archived log file no longer needed for recovery. This file, and all files up to and including *lastarcdel*, can be moved to make room on the disk.
		For example, if the values returned in *firstarcdel* and *lastarcdel* are `S0000001.LOG` and `S0000005.LOG`, the following log files can be moved:
		`S0000001.LOG`
		`S0000002.LOG`
		`S0000003.LOG`
		`S0000004.LOG`
		`S0000005.LOG`

LASTARCDEL	UNSIGNED CHAR(13)	A 12-byte buffer to hold the returned name of the last archived log file that can be removed from the database log directory.

Table 7 (Page 2 of 2). Fields in the RFWD-OUTPUT Structure

Field Name	Data Type	Description
LASTCOMMIT	UNSIGNED CHAR(27)	A 26-character string containing a time stamp in ISO format. This value represents the time stamp of the last committed transaction after the rollforward operation terminates.

Possible values for *STATE* (defined in `sqlutil`) are:

SQLURFQ_NOT_AVAILABLE
Could not connect to the node.

SQLURFQ_NOT_RFW_PENDING
Database is not rollforward pending.

SQLURFQ_DB_RFW_PENDING
Database is rollforward pending.

SQLURFQ_TBL_RFW_PENDING
Table space is rollforward pending.

SQLURFQ_DB_RFW_IN_PROGRESS
Database rollforward in progress.

SQLURFQ_TBL_RFW_IN_PROGRESS
Table space rollforward in progress.

SQLURFQ_DB_RFW_STOPPING
Database rollforward was interrupted while processing a STOP request.

SQLURFQ_TBL_RFW_STOPPING
Table space rollforward was interrupted while processing a STOP request.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: RFWD-OUTPUT */
/* ... */
SQL_STRUCTURE rfwd_output
{
  char            *pApplicationId;
  long            *pNumReplies;
  struct sqlurf_info *pNodeInfo;
};
/* ... */
/* File: sqlutil.h */
/* Structure: SQLURF-INFO */
/* ... */
SQL_STRUCTURE sqlurf_info
{
  SQL_PDB_NODE_TYPE nodenum;
  long              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];
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-RFWD-OUTPUT.
    05 SQL-APPLID             USAGE IS POINTER.
    05 SQL-NUMREPLIES         USAGE IS POINTER.
    05 SQL-P-NODE-INFO        USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQLURF-INFO.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-STATE              PIC S9(9) COMP-5.
    05 SQL-NEXTARCLOG         PIC X(12).
    05 FILLER                 PIC X.
    05 SQL-FIRSTARCDEL        PIC X(12).
    05 FILLER                 PIC X.
    05 SQL-LASTARCDEL         PIC X(12).
    05 FILLER                 PIC X.
    05 SQL-LASTCOMMIT         PIC X(26).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(2).
*
```

SQL-AUTHORIZATIONS

This structure is used to return information after a call to “sqluadau - Get Authorizations” on page 227. The data type of all fields is SMALLINT. The first half of the following table contains authorities granted directly to a user. The second half of the table contains authorities granted to the groups to which a user belongs.

Table 8. Fields in the SQL-AUTHORIZATIONS Structure

Field Name	Description
SQL_AUTHORIZATIONS_LEN	Size of structure.
SQL_SYSADM_AUTH	SYSADM authority.
SQL_SYSCTRL_AUTH	SYSCTRL authority.
SQL_SYSMAINT_AUTH	SYSMAINT authority.
SQL_DBADM_AUTH	DBADM authority.
SQL_CREATETAB_AUTH	CREATETAB authority.
SQL_CREATET_NOT_FENC_AUTH	CREATE_NOT_FENCED authority.
SQL_BINDADD_AUTH	BINDADD authority.
SQL_CONNECT_AUTH	CONNECT authority.
SQL_IMPLICIT_SCHEMA_AUTH	IMPLICIT_SCHEMA authority.
SQL_SYSADM_GRP_AUTH	User belongs to a group which holds SYSADM authority.
SQL_SYSCTRL_GRP_AUTH	User belongs to a group which holds SYSCTRL authority.
SQL_SYSMAINT_GRP_AUTH	User belongs to a group which holds SYSMAINT authority.
SQL_DBADM_GRP_AUTH	User belongs to a group which holds DBADM authority.
SQL_CREATETAB_GRP_AUTH	User belongs to a group which holds CREATETAB authority.
SQL_CREATE_NON_FENC_GRP_AUTH	User belongs to a group which holds CREATE_NOT_FENCED authority.
SQL_BINDADD_GRP_AUTH	User belongs to a group which holds BINDADD authority.
SQL_CONNECT_GRP_AUTH	User belongs to a group which holds CONNECT authority.
SQL_IMPLICIT_SCHEMA_GRP_AUTH	User belongs to a group which holds IMPLICIT_SCHEMA authority.
Note: SYSADM, SYSMAINT, and SYSCTRL are only indirect authorities and cannot be granted directly to the user. They are available only through the groups to which the user belongs.	

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-AUTHORIZATIONS */
/* ... */
SQL_STRUCTURE sql_authorizations
{
  short          sql_authorizations_len;
  short          sql_sysadm_auth;
  short          sql_dbadm_auth;
  short          sql_createtab_auth;
  short          sql_bindadd_auth;
  short          sql_connect_auth;
  short          sql_sysadm_grp_auth;
  short          sql_dbadm_grp_auth;
  short          sql_createtab_grp_auth;
  short          sql_bindadd_grp_auth;
  short          sql_connect_grp_auth;
  short          sql_sysctrl_auth;
  short          sql_sysctrl_grp_auth;
  short          sql_sysmaint_auth;
  short          sql_sysmaint_grp_auth;
  short          sql_create_not_fenc_auth;
  short          sql_create_not_fenc_grp_auth;
  short          sql_implicit_schema_auth;
  short          sql_implicit_schema_grp_auth;
};
/* ... */
```

SQL-AUTHORIZATIONS

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-AUTHORIZATIONS.
    05 SQL-AUTHORIZATIONS-LEN PIC S9(4) COMP-5.
    05 SQL-SYSADM-AUTH        PIC S9(4) COMP-5.
    05 SQL-DBADM-AUTH         PIC S9(4) COMP-5.
    05 SQL-CREATETAB-AUTH     PIC S9(4) COMP-5.
    05 SQL-BINDADD-AUTH       PIC S9(4) COMP-5.
    05 SQL-CONNECT-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSADM-GRP-AUTH    PIC S9(4) COMP-5.
    05 SQL-DBADM-GRP-AUTH     PIC S9(4) COMP-5.
    05 SQL-CREATETAB-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-BINDADD-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-CONNECT-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-AUTH      PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-GRP-AUTH  PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-AUTH PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-GRP-AUTH PIC S9(4) COMP-5.
*
```

SQL-DIR-ENTRY

This structure is used by the DCS directory APIs.

Table 9. Fields in the SQL-DIR-ENTRY Structure

Field Name	Data Type	Description
STRUCT_ID	SMALLINT	Structure identifier. Set to SQL_DCS_STR_ID (defined in sqlenv).
RELEASE	SMALLINT	Release version (assigned by the API).
CODEPAGE	SMALLINT	Code page for comment.
COMMENT	CHAR(30)	Optional description of the database.
LDB	CHAR(8)	Local name of the database; must match database alias in system database directory.
TDB	CHAR(18)	Actual name of the database.
AR	CHAR(32)	Name of the application client.
PARM	CHAR(512)	Contains transaction program prefix, transaction program name, SQLCODE mapping file name, and disconnect and security option.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQL-DIR-ENTRY */
/* ... */
SQL_STRUCTURE sql_dir_entry
{
  unsigned short        struct_id;
  unsigned short        release;
  unsigned short        codepage;
  _SQLOLDCHAR           comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR           ldb[SQL_DBNAME_SZ + 1];
  _SQLOLDCHAR           tdb[SQL_LONG_NAME_SZ + 1];
  _SQLOLDCHAR           ar[SQL_AR_SZ + 1];
  _SQLOLDCHAR           parm[SQL_PARAMETER_SZ + 1];
};
/* ... */
```

SQL-DIR-ENTRY

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-DIR-ENTRY.
    05 STRUCT-ID               PIC 9(4) COMP-5.
    05 RELEASE-LVL             PIC 9(4) COMP-5.
    05 CODEPAGE                PIC 9(4) COMP-5.
    05 COMMENT                 PIC X(30).
    05 FILLER                  PIC X.
    05 LDB                     PIC X(8).
    05 FILLER                  PIC X.
    05 TDB                     PIC X(18).
    05 FILLER                  PIC X.
    05 AR                      PIC X(32).
    05 FILLER                  PIC X.
    05 PARM                    PIC X(512).
    05 FILLER                  PIC X.
    05 FILLER                  PIC X(1).
*
```

SQLA-FLAGINFO

This structure is used to hold flagger information.

Table 10. Fields in the SQLA-FLAGINFO Structure

Field Name	Data Type	Description
VERSION	SMALLINT	Input field that must be set to SQLA_FLAG_VERSION (defined in `sqlaprep`).
MSGS	Structure	An imbedded *sqla_flagmsgs* structure.

Table 11. Fields in the SQLA-FLAGMSGS Structure

Field Name	Data Type	Description
COUNT	SMALLINT	Output field set to the number of messages returned by the flagger.
SQLCA	Array	Array of SQLCA structures returning information from the flagger.

Language Syntax

C Structure

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGINFO */
/* ... */
SQL_STRUCTURE sqla_flaginfo
{
  short             version;
  short             padding;
  struct            sqla_flagmsgs msgs;
};
/* ... */
```

```
/* File: sqlaprep.h */
/* Structure: SQLA-FLAGMSGS */
/* ... */
SQL_STRUCTURE sqla_flagmsgs
{
  short             count;
  short             padding;
  SQL_STRUCTURE sqlca sqlca[SQLA_FLAG_MAXMSGS];
};
/* ... */
```

COBOL Structure

```
* File: sqlaprep.cbl
01 SQLA-FLAGINFO.
    05 SQLFLAG-VERSION            PIC 9(4) COMP-5.
    05 FILLER                     PIC X(2).
    05 SQLFLAG-MSGS.
        10 SQLFLAG-MSGS-COUNT      PIC 9(4) COMP-5.
        10 FILLER                  PIC X(2).
        10 SQLFLAG-MSGS-SQLCA OCCURS 10 TIMES.
*
```

SQLB-TBS-STATS

This structure is used to return additional table space statistics to an application program.

Table 12. Fields in the SQLB-TBS-STATS Structure

Field Name	Data Type	Description
TOTALPAGES	INTEGER	Total operating system space occupied by the table space (in 4KB pages). For DMS, this is the sum of the container sizes (including overhead). For SMS, this is the sum of all file space used for the tables stored in this table space. This is the only piece of information returned for SMS table spaces; the other fields are set to this value or zero.
USEABLEPAGES	INTEGER	For DMS, equal to TOTALPAGES minus (overhead plus partial extents). For SMS, equal to TOTALPAGES.
USEDPAGES	INTEGER	For DMS, the total number of pages in use. For more information, see "Designing and Choosing Table Spaces" in the *Administration Guide*. For SMS, equal to TOTALPAGES.
FREEPAGES	INTEGER	For DMS, equal to USEABLEPAGES minus USEDPAGES. For SMS, not applicable.
HIGHWATERMARK	INTEGER	For DMS, the high water mark is the current "end" of the table space address space. In other words, the page number of the first free extent following the last allocated extent of a table space. Note that this is not really a "high water mark", but rather a "current water mark", since the value can decrease. For SMS, this is not applicable.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLB-TBS-STATS */
/* ... */
SQL_STRUCTURE SQLB_TBS_STATS
{
  unsigned long   totalPages;
  unsigned long   useablePages;
  unsigned long   usedPages;
  unsigned long   freePages;
  unsigned long   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 13. 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).

SQLB-TBSCONTQRY-DATA

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLB-TBSCONTQRY-DATA */
/* ... */
SQL_STRUCTURE SQLB_TBSCONTQRY_DATA
{
  unsigned long   id;
  unsigned long   nTbs;
  unsigned long   tbsID;
  unsigned long   nameLen;
  char            name[SQLB_MAX_CONTAIN_NAME_SZ];
  unsigned long   underDBDir;
  unsigned long   contType;
  unsigned long   totalPages;
  unsigned long   useablePages;
  unsigned long   ok;
};
/* ... */
```

COBOL Structure

```
* File: sqlutbcq.cbl
01 SQLB-TBSCONTQRY-DATA.
    05 SQL-ID                 PIC 9(9) COMP-5.
    05 SQL-N-TBS              PIC 9(9) COMP-5.
    05 SQL-TBS-ID             PIC 9(9) COMP-5.
    05 SQL-NAME-LEN           PIC 9(9) COMP-5.
    05 SQL-NAME               PIC X(256).
    05 SQL-UNDER-DBDIR        PIC 9(9) COMP-5.
    05 SQL-CONT-TYPE          PIC 9(9) COMP-5.
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-OK                 PIC 9(9) COMP-5.
*
```

SQLB-TBSPQRY-DATA

This structure is used to return table space data to an application program.

Table 14. Fields in the SQLB-TBSPQRY-DATA Structure

Field Name	Data Type	Description
TBSPQVER	CHAR(8)	Structure version identifier.
ID	INTEGER	Internal identifier for the table space.
NAMELEN	INTEGER	Length of the table space name.
NAME	CHAR(128)	Null-terminated name of the table space.
TOTALPAGES	INTEGER	Number of pages specified by CREATE TABLESPACE (DMS only).
USEABLEPAGES	INTEGER	TOTALPAGES minus overhead (DMS only). This value is rounded down to the next multiple of 4KB.
FLAGS	INTEGER	Bit attributes for the table space.
PAGESIZE	INTEGER	Page size (in bytes) of the table space. Currently fixed at 4KB.
EXTSIZE	INTEGER	Extent size (in pages) of the table space.
PREFETCHSIZE	INTEGER	Prefetch size.
NCONTAINERS	INTEGER	Number of containers in the table space.
TBSSTATE	INTEGER	Table space states.
LIFELSN	CHAR(6)	Time stamp identifying the origin of the table space.
FLAGS2	INTEGER	Bit attributes for the table space.
MINIMUMRECTIME	CHAR(27)	Earliest point in time that may be specified by point-in-time table space rollforward.
STATECHNGOBJ	INTEGER	If TBSSTATE is SQLB_LOAD_PENDING or SQLB_DELETE_PENDING, the object ID in table space STATECHANGEID that caused the table space state to be set. Otherwise zero.
STATECHNGID	INTEGER	If TBSSTATE is SQLB_LOAD_PENDING or SQLB_DELETE_PENDING, the table space ID of the object STATECHANGEOBJ that caused the table space state to be set. Otherwise zero.
NQUIESCERS	INTEGER	If TBSSTATE is SQLB_QUIESCED_SHARE, UPDATE, or EXCLUSIVE, the number of quiescers of the table space and the number of entries in QUIESCERS.
QUIESCEID	INTEGER	The table space ID of the object QUIESCEOBJ that caused the table space to be quiesced.
QUIESCEOBJ	INTEGER	The object ID in table space QUIESCEID that caused the table space to be quiesced.
RESERVED	CHAR(32)	Reserved for future use.

Possible values for *FLAGS* (defined in `sqlutil`) are:

SQLB_TBS_SMS
System Managed Space

SQLB_TBS_DMS
Database Managed Space

SQLB_TBS_ANY
Regular contents

SQLB_TBS_LONG
Long field data

SQLB_TBS_TMP
Temporary data.

Possible values for *TBSSTATE* (defined in `sqlutil`) are:

SQLB_NORMAL
Normal

SQLB_QUIESCED_SHARE
Quiesced: SHARE

SQLB_QUIESCED_UPDATE
Quiesced: UPDATE

SQLB_QUIESCED_EXCLUSIVE
Quiesced: EXCLUSIVE

SQLB_LOAD_PENDING
Load pending

SQLB_DELETE_PENDING
Delete pending

SQLB_BACKUP_PENDING
Backup pending

SQLB_ROLLFORWARD_IN_PROGRESS
Roll forward in progress

SQLB_ROLLFORWARD_PENDING
Roll forward pending

SQLB_RESTORE_PENDING
Restore pending

SQLB_DISABLE_PENDING
Disable pending

SQLB_REORG_IN_PROGRESS
Reorganization in progress

SQLB_BACKUP_IN_PROGRESS
Backup in progress

SQLB_STORDEF_PENDING
Storage must be defined

SQLB_RESTORE_IN_PROGRESS
Restore in progress

SQLB_STORDEF_ALLOWED
Storage may be defined

SQLB_STORDEF_FINAL_VERSION
Storage definition is in 'final' state

SQLB_STORDEF_CHANGED
Storage definition was changed prior to roll forward

SQLB_REBAL_IN_PROGRESS
DMS rebalancer is active

SQLB_PSTAT_DELETION
Table space deletion in progress
SQLB_PSTAT_CREATION
Table space creation in progress.

Possible values for *FLAGS2* (defined in `sqlutil`) are:

SQLB_STATE_SET
For service use only.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_TBSPQRY_DATA
{
   char                         tbspqverfflSQLB_SVERSION_SIZE";
   unsigned long                id;
   unsigned long                nameLen;
   char                         namefflSQLB_MAX_TBS_NAME_SZ";
   unsigned long                totalPages;
   unsigned long                useablePages;
   unsigned long                flags;
   unsigned long                pageSize;
   unsigned long                extSize;
   unsigned long                prefetchSize;
   unsigned long                nContainers;
   unsigned long                tbsState;
   char                         lifeLSNffl6";
   char                         padffl2";
   unsigned long                flags2;
   char                         minimumRecTimefflSQL_STAMP_STRLEN+1";
   char                         pad1ffl1";
   unsigned long                StateChngObj;
   unsigned long                StateChngID;
   unsigned long                nQuiescers;
   struct SQLB_QUIESCER_DATA    quiescerfflSQLB_MAX_QUIESCERS";
   char                         reservedffl32";
};
/* ... */
/* File: sqlutil.h */
/* ... */
SQL_STRUCTURE SQLB_QUIESCER_DATA
{
   unsigned long   quiesceId;
   unsigned long   quiesceObject;
};
/* ... */
```

COBOL Structure

```
* File: sqlutbsp.cbl
01 SQLB-TBSPQRY-DATA.
    05 SQL-TBSPQVER           PIC X(8).
    05 SQL-ID                 PIC 9(9) COMP-5.
    05 SQL-NAME-LEN           PIC 9(9) COMP-5.
    05 SQL-NAME               PIC X(128).
    05 SQL-TOTAL-PAGES        PIC 9(9) COMP-5.
    05 SQL-USEABLE-PAGES      PIC 9(9) COMP-5.
    05 SQL-FLAGS              PIC 9(9) COMP-5.
    05 SQL-PAGE-SIZE          PIC 9(9) COMP-5.
    05 SQL-EXT-SIZE           PIC 9(9) COMP-5.
    05 SQL-PREFETCH-SIZE      PIC 9(9) COMP-5.
    05 SQL-N-CONTAINERS       PIC 9(9) COMP-5.
    05 SQL-TBS-STATE          PIC 9(9) COMP-5.
    05 SQL-LIFE-LSN           PIC X(6).
    05 SQL-PAD                PIC X(2).
    05 SQL-FLAGS2             PIC 9(9) COMP-5.
    05 SQL-MINIMUM-REC-TIME   PIC X(26).
    05 FILLER                 PIC X.
    05 SQL-PAD1               PIC X(1).
    05 SQL-STATE-CHNG-OBJ     PIC 9(9) COMP-5.
    05 SQL-STATE-CHNG-ID      PIC 9(9) COMP-5.
    05 SQL-N-QUIESCERS        PIC 9(9) COMP-5.
    05 SQL-QUIESCER OCCURS 5 TIMES.
        10 SQL-QUIESCE-ID     PIC 9(9) COMP-5.
        10 SQL-QUIESCE-OBJECT PIC 9(9) COMP-5.
    05 SQL-RESERVED           PIC X(32).
*
```

SQLCA

The SQL Communication 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.

For detailed information about the SQLCA structure, including a description of its fields, see the *SQL Reference*.

Language Syntax

C Structure

```
/* File: sqlca.h */
/* Structure: SQLCA */
/* ... */
SQL_STRUCTURE  sqlca
{
  _SQLOLDCHAR    sqlcaid[8];
  long           sqlcabc;
  #ifdef DB2_SQL92E
  long           sqlcade;
  #else
  long           sqlcode;
  #endif
  short          sqlerrml;
  _SQLOLDCHAR    sqlerrmc[70];
  _SQLOLDCHAR    sqlerrp[8];
  long           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 15. Fields in the SQLCHAR Structure

Field Name	**Data Type**	**Description**
LENGTH	SMALLINT	Length of the character string pointed to by *DATA*.
DATA	CHAR(n)	An array of characters of length *LENGTH*.

Language Syntax

C Structure

```
/* File: sql.h */
/* Structure: SQLCHAR */
/* ... */
SQL_STRUCTURE sqlchar
{
  short                 length;
  _SQLOLDCHAR           data[1];
};
/* ... */
```

COBOL Structure

This is not defined in any header file. The following is an example showing how it can be done:

```
* Replace maxlen with the appropriate value:
01 SQLCHAR.
49 SQLCHAR-LEN  PIC S9(4) COMP-5.
49 SQLCHAR-DATA PIC X(maxlen).
```

SQLDA

The SQL Descriptor Area (SQLDA) structure is a collection of variables that is required for execution of the SQL DESCRIBE statement. The SQLDA variables are options that can be used with the PREPARE, OPEN, FETCH, EXECUTE, and CALL statements.

An SQLDA communicates with dynamic SQL; it can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH statement.

SQLDAs are supported for all languages, but predefined declarations are provided only for C, REXX, FORTRAN, and COBOL. In REXX, the SQLDA is somewhat different than in the other languages; for information about the use of SQLDAs in REXX, see the *Embedded SQL Programming Guide*.

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.

For detailed information about the SQLDA structure, including a description of its fields, see the *SQL Reference*.

Language Syntax

C Structure

```
/* File: sqlda.h */
/* Structure: SQLDA */
/* ... */
SQL_STRUCTURE  sqlda
{
  _SQLOLDCHAR    sqldaid[8];
  long           sqldabc;
  short          sqln;
  short          sqld;
  struct sqlvar  sqlvar[1];
};
/* ... */
/* File: sqlda.h */
/* Structure: SQLVAR */
/* ... */
SQL_STRUCTURE  sqlvar
{
  short          sqltype;
  short          sqllen;
  _SQLOLDCHAR   *SQL_POINTER sqldata;
  short         *SQL_POINTER sqlind;
  struct sqlname sqlname;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLNAME */
/* ... */
SQL_STRUCTURE  sqlname
{
  short          length;
  _SQLOLDCHAR    data[30];
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLVAR2 */
/* ... */
SQL_STRUCTURE  sqlvar2
{
  union sql8bytelen len;
  char *SQL_POINTER sqldatalen;
  struct sqldistinct_type sqldatatype_name;
};
/* ... */
/* File: sqlda.h */
/* Structure: SQL8BYTELEN */
/* ... */
union sql8bytelen
{
  long        reserve1[2];
  long        sqllonglen;
};
/* ... */
```

```
/* File: sqlda.h */
/* Structure: SQLDISTINCT-TYPE */
/* ... */
SQL_STRUCTURE  sqldistinct_type
{
  short          length;
  char           data[27];
  char           reserved1[3];
};
/* ... */
```

COBOL Structure

```
* File: sqlda.cbl
01 SQLDA SYNC.
    05 SQLDAID PIC X(8) VALUE "SQLDA  ".
    05 SQLDABC PIC S9(9) COMP-5.
    05 SQLN PIC S9(4) COMP-5.
    05 SQLD PIC S9(4) COMP-5.
    05 SQLVAR-ENTRIES OCCURS 0 TO 1489 TIMES
        10 SQLVAR.
        10 SQLVAR2 REDEFINES SQLVAR.
*
```

SQLDCOL

This structure is used to pass variable column information to “sqluimpr - Import” on page 271, “sqluexpr - Export” on page 241, and “sqluload - Load” on page 282.

Table 16. Fields in the SQLDCOL Structure

Field Name	Data Type	Description
DCOLMETH	SMALLINT	A character indicating the method to be used to select columns within the data file.
DCOLNUM	SMALLINT	The number of columns specified in the array *DCOLNAME*.
DCOLNAME	Array	An array of *DCOLNUM sqldcoln* structures.

Table 17. Fields in the SQLDCOLN Structure

Field Name	Data Type	Description
DCOLNLEN	SMALLINT	Length of the data pointed to by *DCOLNPTR*.
DCOLNPTR	Pointer	Pointer to a data element determined by *DCOLMETH*.
Note: The *DCOLNLEN* and *DCOLNPTR* fields are repeated for each column specified.		

Table 18. Fields in the SQLLOCTAB Structure

Field Name	Data Type	Description
LOCPAIR	Array	An array of *sqllocpair* structures.

Table 19. Fields in the SQLLOCPAIR Structure

Field Name	Data Type	Description
BEGIN_LOC	SMALLINT	Starting position of the column data in the external file.
END_LOC	SMALLINT	Ending position of the column data in the external file.

The valid values for *DCOLMETH* (defined in `sqlutil`) are:

SQL_METH_N
Names. When importing or loading, use the column names provided via this structure to identify the data to import or load from the external file. The case of these column names must match the case of the corresponding names in the

system catalogs. When exporting, use the column names provided via this structure as the column names in the output file.

The *dcolnptr* pointer of each element of the *dcolname* array points to an array of characters, of length *dcolnlen* bytes, that make up the name of a column to be imported or loaded. The *dcolnum* field, which must be positive, indicates the number of elements in the *dcolname* array.

This method is invalid if the external file does not contain column names (DEL or ASC format files, for example).

SQL_METH_P

Positions. When importing or loading, use starting column positions provided via this structure to identify the data to import or load from the external file. This method is not valid when exporting data.

The *dcolnptr* pointer of each element of the *dcolname* array is ignored, while the *dcolnlen* field contains a column position in the external file. The *dcolnum* field, which must be positive, indicates the number of elements in the *dcolname* array.

The lowest valid column position value is 1 (indicating the first column), and the highest valid value depends on the external file type. Positional selection is not valid for import of ASC files.

SQL_METH_L

Locations. When importing or loading, use starting and ending column positions provided via this structure to identify the data to import or load from the external file. This method is not valid when exporting data.

The *dcolnptr* field of the first element of the *dcolname* array points to an *sqlloctab* structure, which consists of an array of *sqllocpair* structures. The number of elements in this array is determined by the *dcolnum* field of the *sqldcol* structure, which must be positive. Each element in the array is a pair of 2-byte integers that indicate where the column begins and ends. The first element of each location pair is the byte within the file where the column begins, and the second element is the byte where the column ends. The first byte position within a row in the file is considered byte position 1. The columns can overlap.

This method is the only valid method for importing or loading ASC files.

SQL_METH_D

Default. When importing or loading, 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.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLDCOL */
/* ... */
SQL_STRUCTURE sqldcoln
{
  short           dcolnlen;
  char            *dcolnptr;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLDCOLN */
/* ... */
SQL_STRUCTURE sqldcoln
{
  short           dcolnlen;
  char            *dcolnptr;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLLOCTAB */
/* ... */
SQL_STRUCTURE sqlloctab
{
  struct sqllocpair locpair[1];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLLOCPAIR */
/* ... */
SQL_STRUCTURE sqllocpair
{
  short           begin_loc;
  short           end_loc;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-DCOLDATA.
    05 SQL-DCOLMETH           PIC S9(4) COMP-5.
    05 SQL-DCOLNUM            PIC S9(4) COMP-5.
    05 SQLDCOLN OCCURS 0 TO 255 TIMES DEPENDING ON SQL-DCOLNUM.
        10 SQL-DCOLNLEN       PIC S9(4) COMP-5.
        10 FILLER             PIC X(2).
        10 SQL-DCOLN-PTR      USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQL-LOCTAB.
    05 SQL-LOC-PAIR OCCURS 1 TIMES.
        10 SQL-BEGIN-LOC      PIC S9(4) COMP-5.
        10 SQL-END-LOC        PIC S9(4) COMP-5.
*
```

SQLE-ADDN-OPTIONS

This structure is used to pass information to “sqleaddn - Add Node” on page 65.

Table 20. Fields in the SQLE-NODE-APPN Structure

Field Name	Data Type	Description
SQLADDID	CHAR	An "eyecatcher" value which must be set to `SQLE_ADDOPTID_V51`.
TBLSPACE_TYPE	UNSIGNED LONG	Specifies the type of temporary table space definitions to be used for the node being added. See below for values.
TBLSPACE_NODE	SQL_PDB_NODE_TYPE	Specifies the node number from which the temporary table space definitions should be obtained. The node number must exist in the `db2nodes.cfg` file, and is only used if the *tblspace_type* field is set to `SQLE_TABLESPACES_LIKE_NODE`.

Valid values for *TBLSPACE_TYPE* (defind in `sqlenv`) are:

SQLE_TABLESPACES_NONE
Do not create any temporary table spaces.

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

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

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-ADDN-OPTIONS */
/* ... */
SQL_STRUCTURE sqle_addn_options
{
  char                  sqladdid[8];
  unsigned long         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-CONN-SETTING

This structure is used to specify connection setting types and values (see “sqleqryc - Query Client” on page 162, and “sqlesetc - Set Client” on page 176).

Table 21. Fields in the SQLE-CONN-SETTING Structure

Field Name	Data Type	Description
TYPE	SMALLINT	Setting type.
VALUE	SMALLINT	Setting value.

Connection Settings

The valid entries for the SQLE-CONN-SETTING TYPE element and the associated descriptions for each entry are listed below (defined in `sqlenv` and `sql`):

Table 22 (Page 1 of 3). Connection Settings

Type	Value	Description
SQL_CONNECT_TYPE	SQL_CONNECT_1	Type 1 CONNECTs enforce the single database per unit of work semantics of older releases, also known as the rules for remote unit of work (RUOW).
	SQL_CONNECT_2	Type 2 CONNECTs support the multiple databases per unit of work semantics of DUOW.
SQL_RULES	SQL_RULES_DB2	Enable the SQL CONNECT statement to switch the current connection to an established (dormant) connection.
	SQL_RULES_STD	Permit only the establishment of a new connection through the SQL CONNECT statement. The SQL SET CONNECTION statement must be used to switch the current connection to a dormant connection.
SQL_DISCONNECT	SQL_DISCONNECT_EXPL	Removes those connections that have been explicitly marked for release by the SQL RELEASE statement at commit.
	SQL_DISCONNECT_COND	Breaks those connections that have no open WITH HOLD cursors at commit, and those that have been marked for release by the SQL RELEASE statement.
	SQL_DISCONNECT_AUTO	Breaks all connections at commit.
SQL_SYNCPOINT	SQL_SYNC_TWOPHASE	Requires a Transaction Manager (TM) to coordinate two-phase commits among databases that support this protocol.
	SQL_SYNC_ONEPHASE	Uses one-phase commits to commit the work done by each database in multiple database transactions. Enforces single updater, multiple read behavior.

Table 22 (Page 2 of 3). Connection Settings

Type	Value	Description
	SQL_SYNC_NONE	Uses one-phase commits to commit work done, but does not enforce single updater, multiple read behavior.
SQL_MAX_NETBIOS_CONNECTIONS	Between 1 and 254	This specifies the maximum number of concurrent connections that can be made using a NETBIOS adapter in an application.
SQL_DEFERRED_PREPARE	SQL_DEFERRED_PREPARE_NO	The PREPARE statement will be executed at the time it is issued.
	SQL_DEFERRED_PREPARE_YES	Execution of the PREPARE statement will be deferred until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued. The PREPARE statement will not be deferred if it uses the INTO clause, which requires an SQLDA to be returned immediately. However, if the PREPARE INTO statement is issued for a cursor that does not use any parameter markers, the processing will be optimized by pre-OPENing the cursor when the PREPARE is executed.
	SQL_DEFERRED_PREPARE_ALL	Same as YES, except that a PREPARE INTO statement which contains parameter markers *is* deferred. If a PREPARE INTO statement does not contain parameter markers, pre-OPENing of the cursor will still be performed. If the PREPARE statement uses the INTO clause to return an SQLDA, the application must not reference the content of this SQLDA until the OPEN, DESCRIBE, or EXECUTE statement is issued and returned.
SQL_CONNECT_NODE	Between 0 and 999, or the keyword SQL_CONN_CATALOG_NODE.	Specifies the node to which a connect is to be made. Overrides the value of the environment variable **DB2NODE**. For example, if nodes 1, 2, and 3 are defined, the client only needs to be able to access one of these nodes. If only node 1 containing databases has been cataloged, and this parameter is set to 3, the next connect attempt will result in a connection at node 3, after an initial connection at node 1.

Table 22 (Page 3 of 3). Connection Settings

Type	Value	Description
SQL_ATTACH_NODE	Between 0 and 999.	Specifies the node to which an attach is to be made. Overrides the value of the environment variable **DB2NODE**. For example, if nodes 1, 2, and 3 are defined, the client only needs to be able to access one of these nodes. If only node 1 containing databases has been cataloged, and this parameter is set to 3, then the next attach attempt will result in an attachment at node 3, after an initial attachment at node 1.

Note: These field names are defined for the C programming language. There are similar names for FORTRAN and COBOL, which have the same semantics.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-CONN-SETTING */
/* ... */
SQL_STRUCTURE sqle_conn_setting
{
  unsigned short         type;
  unsigned short         value;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-CONN-SETTING.
    05 SQLE-CONN-SETTING-ITEM occurs 5 times.
        10 SQLE-CONN-TYPE  pic s9(4) comp-5.
        10 SQLE-CONN-VALUE pic s9(4) comp-5.
*
```

SQLE-NODE-APPC

This structure is used to catalog APPC nodes (see “sqlectnd - Catalog Node” on page 89).

Table 23. Fields in the SQLE-NODE-APPC Structure

Field Name	Data Type	Description
LOCAL_LU	CHAR(8)	Local_lu name.
PARTNER_LU	CHAR(8)	Alias Partner_lu name.
MODE	CHAR(8)	Mode.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPC */
/* ... */
SQL_STRUCTURE sqle_node_appc
{
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-APPC.
    05 LOCAL-LU             PIC X(8).
    05 FILLER               PIC X.
    05 PARTNER-LU           PIC X(8).
    05 FILLER               PIC X.
    05 TRANS-MODE           PIC X(8).
    05 FILLER               PIC X.
*
```

SQLE-NODE-APPN

This structure is used to catalog APPN nodes (see “sqlectnd - Catalog Node” on page 89).

Table 24. Fields in the SQLE-NODE-APPN Structure

Field Name	Data Type	Description
NETWORKID	CHAR(8)	Network ID.
REMOTE_LU	CHAR(8)	Alias Remote_lu name.
LOCAL_LU	CHAR(8)	Alias Local_lu name.
MODE	CHAR(8)	Mode.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-APPN */
/* ... */
SQL_STRUCTURE sqle_node_appn
{
  _SQLOLDCHAR    networkid[SQL_NETID_SZ + 1];
  _SQLOLDCHAR    remote_lu[SQL_RMTLU_SZ + 1];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ + 1];
  _SQLOLDCHAR    mode[SQL_MODE_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-APPN.
    05 NETWORKID             PIC X(8).
    05 FILLER                PIC X.
    05 REMOTE-LU             PIC X(8).
    05 FILLER                PIC X.
    05 LOCAL-LU              PIC X(8).
    05 FILLER                PIC X.
    05 TRANS-MODE            PIC X(8).
    05 FILLER                PIC X.
*
```

SQLE-NODE-CPIC

This structure is used to catalog CPIC nodes (see “sqlectnd - Catalog Node” on page 89).

Table 25. Fields in the SQLE-NODE-CPIC Structure

Field Name	Data Type	Description
SYM_DEST_NAME	CHAR(8)	Symbolic destination name of remote partner.
SECURITY_TYPE	SMALLINT	Security type.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Valid values for *SECURITY_TYPE* (defined in `sqlenv`) are:

SQL_CPIC_SECURITY_NONE
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-CPIC */
/* ... */
SQL_STRUCTURE sqle_node_cpic
{
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ+1];
  unsigned short security_type;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-CPIC.
    05 SYM-DEST-NAME          PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
    05 SECURITY-TYPE          PIC 9(4) COMP-5.
*
```

SQLE-NODE-IPXSPX

This structure is used to catalog IPX/SPX nodes (see "sqlectnd - Catalog Node" on page 89).

Table 26. Fields in the SQLE-NODE-IPXSPX Structure

Field Name	Data Type	Description
FILESERVER	CHAR(48)	Name of the NetWare file server where the DB2 server instance is registered.
OBJECTNAME	CHAR(48)	The database manager server instance is represented as the object, *objectname*, on the NetWare file server. The server's IPX/SPX internetwork address is stored and retrieved from this object.

Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-IPXSPX */
/* ... */
SQL_STRUCTURE sqle_node_ipxspx
{
  char           fileserver[SQL_FILESERVER_SZ+1];
  char           objectname[SQL_OBJECTNAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-IPXSPX.
    05 SQL-FILESERVER         PIC X(48).
    05 FILLER                 PIC X.
    05 SQL-OBJECTNAME         PIC X(48).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-LOCAL

This structure is used to catalog local nodes (see “sqlectnd - Catalog Node” on page 89).

Table 27. Fields in the SQLE-NODE-LOCAL Structure

Field Name	Data Type	Description
INSTANCE_NAME	CHAR(8)	Name of an instance.

Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-LOCAL */
/* ... */
SQL_STRUCTURE sqle_node_local
{
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-LOCAL.
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-NETB

This structure is used to catalog NetBIOS nodes (see “sqlectnd - Catalog Node” on page 89).

Table 28. Fields in the SQLE-NODE-NETB Structure

Field Name	Data Type	Description
ADAPTER	SMALLINT	Local LAN adapter.
REMOTE_NNAME	CHAR(8)	*Nname* of the remote workstation that is stored in the database manager configuration file on the server instance.

Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NETB */
/* ... */
SQL_STRUCTURE sqle_node_netb
{
  unsigned short adapter;
  _SQLOLDCHAR    remote_nname[SQL_RMTLU_SZ + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-NETB.
    05 ADAPTER                PIC 9(4) COMP-5.
    05 REMOTE-NNAME           PIC X(8).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
*
```

SQLE-NODE-NPIPE

This structure is used to catalog named pipe nodes (see “sqlectnd - Catalog Node” on page 89).

Table 29. Fields in the SQLE-NODE-NPIPE Structure

Field Name	Data Type	Description
COMPUTERNAME	CHAR(15)	Computer name.
INSTANCE_NAME	CHAR(8)	Name of an instance.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-NPIPE */
/* ... */
SQL_STRUCTURE sqle_node_npipe
{
  char           computername[SQL_COMPUTERNAME_SZ+1];
  char           instance_name[SQL_INSTNAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-NPIPE.
    05 COMPUTERNAME           PIC X(15).
    05 FILLER                 PIC X.
    05 INSTANCE-NAME          PIC X(8).
    05 FILLER                 PIC X.
*
```

SQLE-NODE-STRUCT

This structure is used to catalog nodes (see “sqlectnd - Catalog Node” on page 89).

Table 30. Fields in the SQLE-NODE-STRUCT Structure

Field Name	Data Type	Description
STRUCT_ID	SMALLINT	Structure identifier.
CODEPAGE	SMALLINT	Code page for comment.
COMMENT	CHAR(30)	Optional description of the node.
NODENAME	CHAR(8)	Local name for the node where the database is located.
PROTOCOL	CHAR(1)	Communications protocol type.

Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.

Valid values for *PROTOCOL* (defined in `sqlenv`) are:

SQL_PROTOCOL_APPC
SQL_PROTOCOL_APPN
SQL_PROTOCOL_CPIC
SQL_PROTOCOL_IPXSPX
SQL_PROTOCOL_LOCAL
SQL_PROTOCOL_NETB
SQL_PROTOCOL_NPIPE
SQL_PROTOCOL_SOCKS
SQL_PROTOCOL_TCPIP

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-STRUCT */
/* ... */
SQL_STRUCTURE sqle_node_struct
{
  unsigned short struct_id;
  unsigned short codepage;
  _SQLOLDCHAR    comment[SQL_CMT_SZ + 1];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ + 1];
  unsigned char  protocol;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-STRUCT.
    05 STRUCT-ID                PIC 9(4) COMP-5.
    05 CODEPAGE                 PIC 9(4) COMP-5.
    05 COMMENT                  PIC X(30).
    05 FILLER                   PIC X.
    05 NODENAME                 PIC X(8).
    05 FILLER                   PIC X.
    05 PROTOCOL                 PIC X.
    05 FILLER                   PIC X(1).
*
```

SQLE-NODE-TCPIP

This structure is used to catalog TCP/IP nodes (see “sqlectnd - Catalog Node” on page 89).

Note: To catalog a TCP/IP SOCKS node, set the PROTOCOL type in the node directory structure to SQL_PROTOCOL_SOCKS before calling the **sqlectnd** API (see “SQLE-NODE-STRUCT” on page 377).

Table 31. Fields in the SQLE-NODE-TCPIP Structure

Field Name	Data Type	Description
HOSTNAME	CHAR(255)	The name of the TCP/IP host on which the DB2 server instance resides.
SERVICE_NAME	CHAR(14)	TCP/IP service name or associated port number of the DB2 server instance.
Note: The character fields passed in this structure must be null terminated or blank filled up to the length of the field.		

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-NODE-TCPIP */
/* ... */
SQL_STRUCTURE sqle_node_tcpip
{
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ+1];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ+1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQL-NODE-TCPIP.
    05 HOSTNAME              PIC X(255).
    05 FILLER                PIC X.
    05 SERVICE-NAME          PIC X(14).
    05 FILLER                PIC X.
*
```

SQLE-REG-NWBINDERY

This structure is used to register/deregister the DB2 server in/from the bindery on the NetWare file server (see “sqleregs - Register” on page 165, and “sqledreg - Deregister” on page 108).

Table 32. Fields in the SQLE-REG-NWBINDERY Structure

Field Name	Data Type	Description
UID	CHAR(48)	User ID used to log into the NetWare file server.
PSWD	CHAR(128)	Password used to validate the user ID.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-REG-NWBINDERY */
/* ... */
SQL_STRUCTURE sqle_reg_nwbindery
{
  char                   uid[SQL_NW_UID_SZ+1];
  unsigned short         reserved_len_1;
  char                   pswd[SQL_NW_PSWD_SZ+1];
  unsigned short         reserved_len_2;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-REG-NWBINDERY.
    05 SQL-UID                PIC X(48).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
    05 SQL-UID-LEN            PIC 9(4) COMP-5.
    05 SQL-PSWD               PIC X(128).
    05 FILLER                 PIC X.
    05 FILLER                 PIC X(1).
    05 SQL-PSWD-LEN           PIC 9(4) COMP-5.
*
```

SQLE-START-OPTIONS

This structure is used to provide the database manager start-up options.

Table 33 (Page 1 of 2). Fields in the SQLE-START-OPTIONS Structure

Field Name	Data Type	Description
SQLOPTID	CHAR	An "eyecatcher" value which must be set to `SQLE_STARTOPTID_V51`.
ISPROFILE	UNSIGNED LONG	Indicates whether a profile is specified. If this field indicates that a profile is not specified, the file `db2profile` is used.
PROFILE	CHAR(236)	The name of the profile file to be executed at each node to define the DB2 environment (MPP only). This file is executed before the nodes are started. The default value is `db2profile`.
ISNODENUM	UNSIGNED LONG	Indicates whether a node number is specified. If specified, the start command only affects the specified node.
NODENUM	SQL_PDB_NODE_TYPE	Node number.
OPTION	UNSIGNED LONG	Specifies an action. See below for values.
ISHOSTNAME	UNSIGNED LONG	Indicates whether a host name is specified.
HOSTNAME[a]	CHAR(256)	System name.
ISPORT	UNSIGNED LONG	Indicates whether a port number is specified.
PORT[a]	SQL_PDB_PORT_TYPE	Port number.
ISNETNAME	UNSIGNED LONG	Indicates whether a net name is specified.
NETNAME[a]	CHAR(256)	Net name.
TBLSPACE_TYPE	UNSIGNED LONG	Specifies the type of temporary table space definitions to be used for the node being added. See below for values.
TBLSPACE_NODE	SQL_PDB_NODE_TYPE	Specifies the node number from which the temporary table space definitions should be obtained. The node number must exist in the `db2nodes.cfg` file, and is only used if the *tblspace_type* field is set to `SQLE_TABLESPACES_LIKE_NODE`.
ISCOMPUTER	UNSIGNED LONG	Indicates whether a computer name is specified. Valid on OS/2 or the Windows operating system only.

Table 33 (Page 2 of 2). Fields in the SQLE-START-OPTIONS Structure

Field Name	Data Type	Description
COMPUTER	CHAR(16)	Computer name. Valid on OS/2 or the Windows operating system only.
PUSERNAME	CHAR	Logon account user name. Valid on OS/2 or the Windows operating system only.
PPASSWORD	CHAR	Logon account password. Valid on OS/2 or the Windows operating system only.

[a] This field is valid only for the `SQLE_ADDNODE` or the `SQLE_RESTART` value of the *OPTION* field.

Valid values for *OPTION* (defind in `sqlenv`) are:

SQLE_NONE
Issue the normal db2start operation.

SQLE_ADDNODE
Issue the ADD NODE command.

SQLE_RESTART
Issue the RESTART DATABASE command.

SQLE_STANDALONE
Start the node in STANDALONE mode.

For more information about these options, see the *Command Reference*.

Valid values for *TBLSPACE_TYPE* (defind in `sqlenv`) are:

SQLE_TABLESPACES_NONE
Do not create any temporary table spaces.

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

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

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLE-START-OPTIONS */
/* ... */
SQL_STRUCTURE sqle_start_options
{
  char                    sqloptid[8];
  unsigned long           isprofile;
  char                    profile[SQL_PROFILE_SZ+1];
  unsigned long           isnodenum;
  SQL_PDB_NODE_TYPE       nodenum;
  unsigned long           option;
  unsigned long           ishostname;
  char                    hostname[SQL_HOSTNAME_SZ+1];
  unsigned long           isport;
  SQL_PDB_PORT_TYPE       port;
  unsigned long           isnetname;
  char                    netname[SQL_HOSTNAME_SZ+1];
  unsigned long           tblspace_type;
  SQL_PDB_NODE_TYPE       tblspace_node;
  unsigned long           iscomputer;
  char                    computer[SQL_COMPUTERNAME_SZ+1];
  char                    *pUserName;
  char                    *pPassword;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLE-START-OPTIONS.
    05 SQLOPTID                 PIC X(8).
    05 SQL-ISPROFILE            PIC 9(9) COMP-5.
    05 SQL-PROFILE              PIC X(235).
    05 FILLER                   PIC X.
    05 SQL-ISNODENUM            PIC 9(9) COMP-5.
    05 SQL-NODENUM              PIC S9(4) COMP-5.
    05 FILLER                   PIC X(2).
    05 SQL-OPTION               PIC 9(9) COMP-5.
    05 SQL-ISHOSTNAME           PIC 9(9) COMP-5.
    05 SQL-HOSTNAME             PIC X(255).
    05 FILLER                   PIC X.
    05 SQL-ISPORT               PIC 9(9) COMP-5.
    05 SQL-PORT                 PIC S9(9) COMP-5.
    05 SQL-ISNETNAME            PIC 9(9) COMP-5.
    05 SQL-NETNAME              PIC X(255).
    05 FILLER                   PIC X.
    05 SQL-TBLSPACE-TYPE        PIC 9(9) COMP-5.
    05 SQL-TBLSPACE-NODE        PIC S9(4) COMP-5.
    05 FILLER                   PIC X(2).
    05 SQL-ISCOMPUTER           PIC 9(9) COMP-5.
    05 SQL-COMPUTER             PIC X(15).
    05 FILLER                   PIC X.
    05 SQL-P-USER-NAME          USAGE IS POINTER.
    05 SQL-P-PASSWORD           USAGE IS POINTER.
*
```

SQLEDBCOUNTRYINFO

This structure is used to provide codeset and territory options to “sqlecrea - Create Database” on page 81.

Table 34. Fields in the SQLEDBCOUNTRYINFO Structure

Field Name	Data Type	Description
SQLDBCODESET	CHAR(9)	Database codeset.
SQLDBLOCALE	CHAR(5)	Database territory.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDBCOUNTRYINFO */
/* ... */
SQL_STRUCTURE sqledbcountryinfo
{
  char                  sqldbcodeset[SQL_CODESET_LEN + 1];
  char                  sqldblocale[SQL_LOCALE_LEN + 1];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDBCOUNTRYINFO.
    05 SQLDBCODESET           PIC X(9).
    05 FILLER                 PIC X.
    05 SQLDBLOCALE            PIC X(5).
    05 FILLER                 PIC X.
*
```

SQLEDBDESC

The Database Description Block (SQLEDBDESC) structure can be used during a call to "sqlecrea - Create Database" on page 81 to specify permanent values for database attributes. These attributes include database comment, collating sequences, and table space definitions.

Table 35. Fields in the SQLEDBDESC Structure

Field Name	Data Type	Description
SQLDBDID	CHAR(8)	A structure identifier and "eye-catcher" for storage dumps. It is a string of eight bytes that must be initialized with the value of `SQLE_DBDESC_2` (defined in `sqlenv`). The contents of this field are validated for version control.
SQLDBCCP	INTEGER	The code page of the database comment. This value is no longer used by the database manager.
SQLDBCSS	INTEGER	A value indicating the source of the database collating sequence.
SQLDBUDC	CHAR(256)	The *n*th byte of this field contains the sort weight of the code point whose underlying decimal representation is *n* in the code page of the database. If SQLDBCSS is not equal to `SQL_CS_USER`, this field is ignored.
SQLDBCMT	CHAR(30)	The comment for the database.
SQLDBSGP	INTEGER	Reserved field. No longer used.
SQLDBNSG	SHORT	A value which indicates the number of file segments to be created in the database. The minimum value for this field is 1 and the maximum value for this field is 256. If a value of -1 is supplied, this field will default to 1. **Note:** SQLDBNSG set to zero produces a default for Version 1 compatibility.
SQLTSEXT	INTEGER	A value, in 4KB pages, which indicates the default extent size for each table space in the database. The minimum value for this field is 2 and the maximum value for this field is 256. If a value of -1 is supplied, this field will default to 32.
SQLCATTS	Pointer	A pointer to a table space description control block, SQLETSDESC, which defines the catalog table space. If null, a default catalog table space based on the values of SQLTSEXT and SQLDBNSG will be created.
SQLUSRTS	Pointer	A pointer to a table space description control block, SQLETSDESC, which defines the user table space. If null, a default user table space based on the values of SQLTSEXT and SQLDBNSG will be created.
SQLTMPTS	Pointer	A pointer to a table space description control block, SQLETSDESC, which defines the temporary table space. If null, a default temporary table space based on the values of SQLTSEXT and SQLDBNSG will be created.

The Tablespace Description Block structure (SQLETSDESC) is used to specify the attributes of any of the three initial table spaces.

Table 36. Fields in the SQLETSDESC Structure		
Field Name	**Data Type**	**Description**
SQLTSDID	CHAR(8)	A structure identifier and "eye-catcher" for storage dumps. It is a string of eight bytes that must be initialized with the value of `SQLE_DBTSDESC_1` (defined in `sqlenv`). The contents of this field are validated for version control.
SQLEXTNT	INTEGER	Table space extentsize, in 4KB pages. If a value of -1 is supplied, this field will default to the current value of the *dft_extent_sz* configuration parameter.
SQLPRFTC	INTEGER	Table space prefetchsize, in 4KB pages. If a value of -1 is supplied, this field will default to the current value of the *dft_prefetch_sz* configuration parameter.
SQLPOVHD	DOUBLE	Table space I/O overhead, in milliseconds. If a value of -1 is supplied, this field will default to an internal database manager value (currently 24.1 ms) that could change with future releases.
SQLTRFRT	DOUBLE	Table space I/O transfer rate, in milliseconds. If a value of -1 is supplied, this field will default to an internal database manager value (currently 0.9 ms) that could change with future releases.
SQLTSTYP	CHAR(1)	Indicates whether the table space is system-managed or database-managed.
SQLCCNT	SMALLINT	Number of containers being assigned to the table space. Indicates how many SQLCTYPE/SQLCSIZE/SQLCLEN/SQLCONTR values follow.
CONTAINR	Array	An array of *sqlccnt SQLETSCDESC* structures.

Table 37. Fields in the SQLETSCDESC Structure		
Field Name	**Data Type**	**Description**
SQLCTYPE	CHAR(1)	Identifies the type of this container.
SQLCSIZE	INTEGER	Size of the container identified in *SQLCONTR,* specified in 4KB pages. Valid only when *SQLTSTYP* is set to `SQL_TBS_TYP_DMS`.
SQLCLEN	SMALLINT	Length of following *SQLCONTR* value.
SQLCONTR	CHAR(256)	Container string.

Valid values for *SQLDBCSS* (defined in `sqlenv`) are:

SQL_CS_SYSTEM
Collating sequence from system.

SQL_CS_USER
Collating sequence from user.

SQL_CS_NONE
None.

SQLE_CS_COMPATABILITY
Use pre-Version 5 collating sequence.

Valid values for *SQLTSTYPE* (defined in `sqlenv`) are:

SQL_TBS_TYP_SMS
System managed

SQL_TBS_TYP_DMS
Database managed.

Valid values for *SQLCTYPE* (defined in `sqlenv`) are:

SQL_TBSC_TYP_DEV
Device. Valid only when *SQLTSTYP* = `SQL_TBS_TYP_DMS`.

SQL_TBSC_TYP_FILE
File. Valid only when *SQLTSTYP* = `SQL_TBS_TYP_DMS`.

SQL_TBSC_TYP_PATH
Path (directory). Valid only when *SQLTSTYP* = `SQL_TBS_TYP_SMS`.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDBDESC */
/* ... */
SQL_STRUCTURE sqledbdesc
{
  _SQLOLDCHAR    sqldbdid[8];
  long           sqldbccp;
  long           sqldbcss;
  unsigned char  sqldbudc[SQL_CS_SZ];
  _SQLOLDCHAR    sqldbcmt[SQL_CMT_SZ+1];
  _SQLOLDCHAR    pad[1];
  unsigned long  sqldbsgp;
  short          sqldbnsg;
  char           pad2[2];
  long           sqltsext;
  struct SQLETSDESC *sqlcatts;
  struct SQLETSDESC *sqlusrts;
  struct SQLETSDESC *sqltmpts;
};
/* ... */
/* File: sqlenv.h */
/* Structure: SQLETSDESC */
/* ... */
SQL_STRUCTURE SQLETSDESC
{
  char           sqltsdid[8];
  long           sqlextnt;
  long           sqlprftc;
  double         sqlpovhd;
  double         sqltrfrt;
  char           sqltstyp;
  char           pad1;
  short          sqlccnt;
  struct SQLETSCDESC containr[1];
};
/* ... */
```

```
/* File: sqlenv.h */
/* Structure: SQLETSCDESC */
/* ... */
SQL_STRUCTURE SQLETSCDESC
{
  char           sqlctype;
  char           pad1[3];
  long           sqlcsize;
  short          sqlclen;
  char           sqlcontr[SQLB_MAX_CONTAIN_NAME_SZ];
  char           pad2[2];
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDBDESC.
    05 SQLDBDID                PIC X(8).
    05 SQLDBCCP                PIC S9(9) COMP-5.
    05 SQLDBCSS                PIC S9(9) COMP-5.
    05 SQLDBUDC                PIC X(256).
    05 SQLDBCMT                PIC X(30).
    05 FILLER                  PIC X.
    05 SQL-PAD                 PIC X(1).
    05 SQLDBSGP                PIC 9(9) COMP-5.
    05 SQLDBNSG                PIC S9(4) COMP-5.
    05 SQL-PAD2                PIC X(2).
    05 SQLTSEXT                PIC S9(9) COMP-5.
    05 SQLCATTS                USAGE IS POINTER.
    05 SQLUSRTS                USAGE IS POINTER.
    05 SQLTMPTS                USAGE IS POINTER.
*
```

```
* File: sqletsd.cbl
01 SQLETSDESC.
    05 SQLTSDID                PIC X(8).
    05 SQLEXTNT                PIC S9(9) COMP-5.
    05 SQLPRFTC                PIC S9(9) COMP-5.
    05 SQLPOVHD                USAGE COMP-2.
    05 SQLTRFRT                USAGE COMP-2.
    05 SQLTSTYP                PIC X.
    05 SQL-PAD1                PIC X.
    05 SQLCCNT                 PIC S9(4) COMP-5.
    05 SQL-CONTAINR OCCURS 001 TIMES.
        10 SQLCTYPE            PIC X.
        10 SQL-PAD1            PIC X(3).
        10 SQLCSIZE            PIC S9(9) COMP-5.
        10 SQLCLEN             PIC S9(4) COMP-5.
        10 SQLCONTR            PIC X(256).
        10 SQL-PAD2            PIC X(2).
*
```

```
* File: sqlenv.cbl
01 SQLETSCDESC.
    05 SQLCTYPE                PIC X.
    05 SQL-PAD1                PIC X(3).
    05 SQLCSIZE                PIC S9(9) COMP-5.
    05 SQLCLEN                 PIC S9(4) COMP-5.
    05 SQLCONTR                PIC X(256).
    05 SQL-PAD2                PIC X(2).
*
```

SQLEDBSTOPOPT

This structure is used to provide the database manager stop options.

Table 38. Fields in the SQLEDBSTOPOPT Structure

Field Name	Data Type	Description
ISPROFILE	UNSIGNED LONG	Indicates whether a profile is specified. If this field indicates that a profile is not specified, the file `db2profile` is used.
PROFILE	CHAR(236)	The name of the profile file that was executed at startup to define the DB2 environment for those nodes that were started (MPP only). If a profile for "sqlepstart - Start Database Manager" on page 156 was specified, the same profile must be specified here.
ISNODENUM	UNSIGNED LONG	Indicates whether a node number is specified. If specified, the start command only affects the specified node.
NODENUM	SQL_PDB_NODE_TYPE	Node number.
OPTION	UNSIGNED LONG	Option.
CALLERAC	UNSIGNED LONG	Caller action. This field is valid only for the SQLE_DROP value of the OPTION field.

Valid values for *OPTION* (defind in `sqlenv`) are:

SQLE_NONE
Issue the normal db2stop operation.

SQLE_FORCE
Issue the FORCE APPLICATION (ALL) command.

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

For more information about these options, see the *Command Reference*.

Valid values for *CALLERAC* (defind in `sqlenv`) are:

SQLE_DROP
Initial call. This is the default value.

SQLE_CONTINUE
Subsequent call. Continue processing after a prompt.

SQLE_TERMINATE
Subsequent call. Terminate processing after a prompt.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDBSTOPOPT */
/* ... */
SQL_STRUCTURE sqledbstopopt
{
  unsigned long          isprofile;
  char                   profile[SQL_PROFILE_SZ+1];
  unsigned long          isnodenum;
  SQL_PDB_NODE_TYPE      nodenum;
  unsigned long          option;
  unsigned long          callerac;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDBSTOPOPT.
    05 SQL-ISPROFILE          PIC 9(9) COMP-5.
    05 SQL-PROFILE            PIC X(235).
    05 FILLER                 PIC X.
    05 SQL-ISNODENUM          PIC 9(9) COMP-5.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-OPTION             PIC 9(9) COMP-5.
    05 SQL-CALLERAC           PIC 9(9) COMP-5.
*
```

SQLEDINFO

This structure is used to return information after a call to “sqledgne - Get Next Database Directory Entry” on page 100. It is shared by both the system database directory and the local database directory.

Table 39. Fields in the SQLEDINFO Structure

Field Name	Data Type	Description
ALIAS	CHAR(8)	An alternate database name.
DBNAME	CHAR(8)	The name of the database.
DRIVE	CHAR(215)	The local database directory path name where the database resides. This field is returned only if the system database directory is opened for scan. **Note:** On OS/2, this field is CHAR(2); on Windows NT, it is CHAR(12).
INTNAME	CHAR(8)	A token identifying the database subdirectory. This field is returned only if the local database directory is opened for scan.
NODENAME	CHAR(8)	The name of the node where the database is located. This field is returned only if the cataloged database is a remote database.
DBTYPE	CHAR(20)	Database manager release information.
COMMENT	CHAR(30)	The comment associated with the database.
COM_CODEPAGE	SMALLINT	The code page of the comment. Not used.
TYPE	CHAR(1)	Entry type. See below for values.
AUTHENTICATION	SMALLINT	Authentication type. See below for values.
GLBDBNAME	CHAR(255)	The global name of the target database in the global (DCE) directory, if the entry is of type `SQL_DCE`.
DCEPRINCIPAL	CHAR(1024)	The DCE principal name if the authentication is of type DCE.
CAT_NODENUM	SHORT	Catalog node number.
NODENUM	SHORT	Node number.

Note: Both system and local database directory use the same structure, but only certain fields are valid for each. Each character field returned is blank filled up to the length of the field.

Valid values for *TYPE* (defind in `sqlenv`) are:

SQL_INDIRECT
Database created by the current instance (as defined by the value of the **DB2INSTANCE** environment variable).

SQL_REMOTE
Database resides at a different instance.

SQL_HOME
Database resides on this volume (always HOME in local database directory).

SQL_DCE
Database resides in DCE directories.

Valid values for *AUTHENTICATION* (defined in `sqlenv`) are:

SQL_AUTHENTICATION_SERVER
Authentication of the user name and password takes place at the server.

SQL_AUTHENTICATION_CLIENT
Authentication of the user name and password takes place at the client.

SQL_AUTHENTICATION_DCS
Used for DDCS.

SQL_AUTHENTICATION_DCE
Authentication takes place using DCE Security Services.

SQL_AUTHENTICATION_NOT_SPECIFIED
DB2 no longer requires authentication to be kept in the database directory. Specify this value when connecting to anything other than a down-level (DB2 V2 or less) server.

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLEDINFO */
/* ... */
SQL_STRUCTURE sqledinfo
{
  _SQLOLDCHAR    alias[SQL_ALIAS_SZ];
  _SQLOLDCHAR    dbname[SQL_DBNAME_SZ];
  _SQLOLDCHAR    drive[SQL_DRIVE_SZ];
  _SQLOLDCHAR    intname[SQL_INAME_SZ];
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    dbtype[SQL_DBTYP_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  short          com_codepage;
  _SQLOLDCHAR    type;
  unsigned short authentication;
  char           glbdbname[SQL_DIR_NAME_SZ];
  _SQLOLDCHAR    dceprincipal[SQL_DCEPRIN_SZ];
  short          cat_nodenum;
  short          nodenum;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLEDINFO.
    05 SQL-ALIAS              PIC X(8).
    05 SQL-DBNAME             PIC X(8).
    05 SQL-DRIVE              PIC X(215).
    05 SQL-INTNAME            PIC X(8).
    05 SQL-NODENAME           PIC X(8).
    05 SQL-DBTYPE             PIC X(20).
    05 SQL-COMMENT            PIC X(30).
    05 FILLER                 PIC X(1).
    05 SQL-COM-CODEPAGE       PIC S9(4) COMP-5.
    05 SQL-TYPE               PIC X.
    05 FILLER                 PIC X(1).
    05 SQL-AUTHENTICATION     PIC 9(4) COMP-5.
    05 SQL-GLBDBNAME          PIC X(255).
    05 SQL-DCEPRINCIPAL       PIC X(1024).
    05 FILLER                 PIC X(1).
    05 SQL-CAT-NODENUM        PIC S9(4) COMP-5.
    05 SQL-NODENUM            PIC S9(4) COMP-5.
*
```

SQLENINFO

This structure returns information after a call to “sqlengne - Get Next Node Directory Entry” on page 150.

Table 40. Fields in the SQLENINFO Structure

Field Name	Data Type	Description
NODENAME	CHAR(8)	Used for the NetBIOS protocol; the *nname* of the node where the database is located (valid in system directory only).
LOCAL_LU	CHAR(8)	Used for the APPN protocol; local logical unit.
PARTNER_LU	CHAR(8)	Used for the APPN protocol; partner logical unit.
MODE	CHAR(8)	Used for the APPN protocol; transmission service mode.
COMMENT	CHAR(30)	The comment associated with the node.
COM_CODEPAGE	SMALLINT	The code page of the comment. This field is no longer used by the database manager.
ADAPTER	SMALLINT	Used for the NetBIOS protocol; the local network adapter.
NETWORKID	CHAR(8)	Used for the APPN protocol; network ID.
PROTOCOL	CHAR(1)	Communications protocol.
SYM_DEST_NAME	CHAR(8)	Used for the APPC protocol; the symbolic destination name.
SECURITY_TYPE	SMALLINT	Used for the APPC protocol; the security type. See below for values.
HOSTNAME	CHAR(255)	Used for the TCP/IP protocol; the name of the TCP/IP host on which the DB2 server instance resides.
SERVICE_NAME	CHAR(14)	Used for the TCP/IP protocol; the TCP/IP service name or associated port number of the DB2 server instance.
FILESERVER	CHAR(48)	Used for the IPX/SPX protocol; the name of the NetWare file server where the DB2 server instance is registered.
OBJECTNAME	CHAR(48)	The database manager server instance is represented as the object, *objectname*, on the NetWare file server. The server's IPX/SPX internetwork address is stored and retrieved from this object.
INSTANCE_NAME	CHAR(8)	Used for the local and NPIPE protocols; the name of the server instance.
COMPUTERNAME	CHAR(15)	Used by the NPIPE protocol; the server node's computer name.
SYSTEM_NAME	CHAR(21)	The DB2 system name of the remote server.
REMOTE_INSTNAME	CHAR(8)	The name of the DB2 server instance.
CATALOG_NODE_TYPE	CHAR	Catalog node type.
OS_TYPE	UNSIGNED SHORT	Identifies the operating system of the server.
Note: Each character field returned is blank filled up to the length of the field.		

Valid values for *SECURITY_TYPE* (defined in `sqlenv`) are:

SQL_CPIC_SECURITY_NONE
SQL_CPIC_SECURITY_SAME
SQL_CPIC_SECURITY_PROGRAM

Language Syntax

C Structure

```
/* File: sqlenv.h */
/* Structure: SQLENINFO */
/* ... */
SQL_STRUCTURE sqleninfo
{
  _SQLOLDCHAR    nodename[SQL_NNAME_SZ];
  _SQLOLDCHAR    local_lu[SQL_LOCLU_SZ];
  _SQLOLDCHAR    partner_lu[SQL_RMTLU_SZ];
  _SQLOLDCHAR    mode[SQL_MODE_SZ];
  _SQLOLDCHAR    comment[SQL_CMT_SZ];
  unsigned short com_codepage;
  unsigned short adapter;
  _SQLOLDCHAR    networkid[SQL_NETID_SZ];
  _SQLOLDCHAR    protocol;
  _SQLOLDCHAR    sym_dest_name[SQL_SYM_DEST_NAME_SZ];
  unsigned short security_type;
  _SQLOLDCHAR    hostname[SQL_HOSTNAME_SZ];
  _SQLOLDCHAR    service_name[SQL_SERVICE_NAME_SZ];
  char           fileserver[SQL_FILESERVER_SZ];
  char           objectname[SQL_OBJECTNAME_SZ];
  char           instance_name[SQL_INSTNAME_SZ];
  char           computername[SQL_COMPUTERNAME_SZ];
  char           system_name[SQL_SYSTEM_NAME_SZ];
  char           remote_instname[SQL_REMOTE_INSTNAME_SZ];
  _SQLOLDCHAR    catalog_node_type;
  unsigned short os_type;
};
/* ... */
```

COBOL Structure

```
* File: sqlenv.cbl
01 SQLENINFO.
    05 SQL-NODE-NAME          PIC X(8).
    05 SQL-LOCAL-LU           PIC X(8).
    05 SQL-PARTNER-LU         PIC X(8).
    05 SQL-MODE               PIC X(8).
    05 SQL-COMMENT            PIC X(30).
    05 SQL-COM-CODEPAGE       PIC 9(4) COMP-5.
    05 SQL-ADAPTER            PIC 9(4) COMP-5.
    05 SQL-NETWORKID          PIC X(8).
    05 SQL-PROTOCOL           PIC X.
    05 SQL-SYM-DEST-NAME      PIC X(8).
    05 FILLER                 PIC X(1).
    05 SQL-SECURITY-TYPE      PIC 9(4) COMP-5.
    05 SQL-HOSTNAME           PIC X(255).
    05 SQL-SERVICE-NAME       PIC X(14).
    05 SQL-FILESERVER         PIC X(48).
    05 SQL-OBJECTNAME         PIC X(48).
    05 SQL-INSTANCE-NAME      PIC X(8).
    05 SQL-COMPUTERNAME       PIC X(15).
    05 SQL-SYSTEM-NAME        PIC X(21).
    05 SQL-REMOTE-INSTNAME    PIC X(8).
    05 SQL-CATALOG-NODE-TYPE  PIC X.
    05 SQL-OS-TYPE            PIC 9(4) COMP-5.
*
```

SQLFUPD

This structure passes information about database configuration files and the database manager configuration file. It is used with the database configuration and database manager configuration APIs.

Table 41. Fields in the SQLFUPD Structure

Field Name	Data Type	Description
TOKEN	UINT16	Specifies the configuration value to return or update.
PTRVALUE	Pointer	A pointer to an application allocated buffer that holds the data specified by *TOKEN*.

Valid data types for the *token* element are:

Uint16	Unsigned 2-byte integer
Sint16	Signed 2-byte integer
Uint32	Unsigned 4-byte integer
Sint32	Signed 4-byte integer
float	4-byte floating-point decimal
char(*n*)	String of length *n* (not including null termination).

Database Configuration File Entries

For a brief description of the database configuration parameters, see “sqlfxdb - Get Database Configuration” on page 201. For more information about these parameters, see the *Administration Guide*. Valid entries for the SQLFUPD *token* element are listed below:

Table 42 (Page 1 of 3). Updateable Database Configuration Parameters

Parameter Name	Token	Token Value	Data Type
adsm_mgmtclass	SQLF_DBTN_ADSM_MGMTCLASS	307	char(30)
adsm_nodename	SQLF_DBTN_ADSM_NODENAME	306	char(64)
adsm_owner	SQLF_DBTN_ADSM_OWNER	305	char(64)
adsm_password	SQLF_DBTN_ADSM_PASSWORD	501	char(64)
app_ctl_heap_sz	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16
applheapsz	SQLF_DBTN_APPLHEAPSZ	51	Uint16
autorestart	SQLF_DBTN_AUTO_RESTART	25	Uint16
avg_appls	SQLF_DBTN_AVG_APPLS	47	Uint16
buffpage	SQLF_DBTN_BUFF_PAGE	90	Uint32
catalogcache_sz	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32
chngpgs_thresh	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16
copyprotect	SQLF_DBTN_COPY_PROTECT	22	Uint16
dbheap	SQLF_DBTN_DBHEAP	50	Uint16
dft_degree	SQLF_DBTN_DFT_DEGREE	301	Sint32

Table 42 (Page 2 of 3). Updateable Database Configuration Parameters

Parameter Name	Token	Token Value	Data Type
dft_extent_sz	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32
dft_loadrec_ses	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16
dft_prefetch_sz	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16
dft_queryopt	SQLF_DBTN_DFT_QUERYOPT	57	Sint32
dft_sqlmathwarn	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16
dir_obj_name	SQLF_DBTN_DIR_OBJ_NAME	46	char(255)
discover	SQLF_DBTN_DISCOVER	308	Uint16
dlchktime	SQLF_DBTN_DLCHKTIME	9	Uint32
estore_seg_sz	SQLF_DBTN_ESTORE_SEG_SZ	303	Sint32
indexrec	SQLF_DBTN_INDEXREC	30	Uint16
indexsort	SQLF_DBTN_INDEXSORT	35	Uint16
locklist	SQLF_DBTN_LOCKLIST	1	Uint16
locktimeout	SQLF_DBTN_LOCKTIMEOUT	34	Sint16
logbufsz	SQLF_DBTN_LOGBUFSZ	33	Uint16
logfilsiz	SQLF_DBTN_LOGFIL_SIZ	92	Uint32
logprimary	SQLF_DBTN_LOGPRIMARY	16	Uint16
logretain	SQLF_DBTN_LOG_RETAIN	23	Uint16
logsecond	SQLF_DBTN_LOGSECOND	17	Uint16
maxappls	SQLF_DBTN_MAXAPPLS	6	Uint16
maxfilop	SQLF_DBTN_MAXFILOP	3	Uint16
maxlocks	SQLF_DBTN_MAXLOCKS	15	Uint16
mincommit	SQLF_DBTN_MINCOMMIT	32	Uint16
newlogpath	SQLF_DBTN_NEWLOGPATH	20	char(242)
num_estore_segs	SQLF_DBTN_NUM_ESTORE_SEGS	304	Sint32
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_quantiles	SQLF_DBTN_NUM_QUANTILES	48	Uint16
pckcachesz	SQLF_DBTN_PCKCACHE_SZ	505	Uint32
rec_his_retentn	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16
seqdetect	SQLF_DBTN_SEQDETECT	41	Uint16
softmax	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
userexit	SQLF_DBTN_USER_EXIT	24	Uint16
util_heap_sz	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32
	SQLF_DBTN_DETS[a]	21	Uint16

Table 42 (Page 3 of 3). Updateable Database Configuration Parameters

Parameter Name	Token	Token Value	Data Type

[a] SQLF_DBTN_DETS is a Uint16 composite parameter, the bits of which indicate database attributes. This allows for the specification of a number of parameters at once. The tokens defining the bits that make up this composite parameter are:

```
Bit SQLF_COPY_PROTECT        (xxx1) : copyprotect
Bit SQLF_ENABLE_LOG_RETAIN   (xx1x) : logretain
Bit SQLF_ENABLE_USER_EXIT    (x1xx) : userexit
Bit SQLF_ENABLE_AUTO_RESTART (1xxx) : autorestart
```

Table 43 (Page 1 of 2). Non-updateable 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)[a]
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)
country	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
nextactive	SQLF_DBTN_NEXTACTIVE	107	char(12)
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)[b]
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16
	SQLF_DBTN_INTFLAGS[a]	104	Uint16

Table 43 (Page 2 of 2). Non-updateable Database Configuration Parameters			
Parameter Name	**Token**	**Token Value**	**Data Type**

[a] SQLF_DBTN_INTFLAGS is a Uint16 parameter, the bits of which indicate database status. This allows for the specification of a number of parameters at once. The tokens defining the bits that make up this composite parameter are:

```
Bit SQLF_CONSISTENT      (xxxx xxx1): database_consistent
Bit SQLF_BACKUP_PENDING  (xxxx x1xx): backup_pending
Bit SQLF_LOG_RETAIN      (xxx1 xxxx): log_retain_status
Bit SQLF_USER_EXIT       (xx1x xxxx): user_exit_status
Bit SQLF_RESTORE_PENDING (1xxx xxxx): restore_pending

The combination of the following two bits:

Bit SQLF_ROLLFWD_PENDING (xxxx 1xxx)
Bit SQLF_TBS_ROLLFWD     (x1xx xxxx)

makes up the rollfwd_pending parameter.
If the SQLF_ROLLFWD_PENDING bit is on, the database requires
rolling forward (rollfwd_pending = SQLF_ENABLE = 1).
If the SQLF_ROLLFWD_PENDING bit is off, and the
SQLF_TBS_ROLLFWD bit is on, one or more table spaces need
to be rolled forward (rollfwd_pending =
SQLF_ROLLFWD_TABLESPACE = 2).
If both bits are off, roll-forward is not pending
(rollfwd_pending = SQLF_DISABLE = 0).
```

[a] char(17) on HP-UX and Solaris.

[b] char(33) on HP-UX and Solaris.

Database Manager Configuration File Entries

For a brief description of the database manager configuration parameters, see “sqlfxsys - Get Database Manager Configuration” on page 204. For more information about these parameters, see the *Administration Guide*. Valid entries for the SQLFUPD *token* element are listed below:

Table 44 (Page 1 of 3). Updateable Database Manager Configuration Parameters

Parameter Name	**Token**	**Token Value**	**Data Type**
agent_stack_sz	SQLF_KTN_AGENT_STACK_SZ	61	Uint16
agentpri	SQLF_KTN_AGENTPRI	26	Sint16
aslheapsz	SQLF_KTN_ASLHEAPSZ	15	Uint32
authentication	SQLF_KTN_AUTHENTICATION	78	Uint16
backbufsz	SQLF_KTN_BACKBUFSZ	18	Uint32
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_client_adpt	SQLF_KTN_DFT_CLIENT_ADPT	82	Uint16
dft_client_comm	SQLF_KTN_DFT_CLIENT_COMM	77	char(31)
dft_monswitches	SQLF_KTN_DFT_MONSWITCHES[a]	29	Uint16

Table 44 (Page 2 of 3). Updateable Database Manager Configuration Parameters

Parameter Name	Token	Token Value	Data Type
dft_mon_bufpool	SQLF_KTN_DFT_MON_BUFPOOL	33	Uint16
dft_mon_lock	SQLF_KTN_DFT_MON_LOCK	34	Uint16
dft_mon_sort	SQLF_KTN_DFT_MON_SORT	35	Uint16
dft_mon_stmt	SQLF_KTN_DFT_MON_STMT	31	Uint16
dft_mon_table	SQLF_KTN_DFT_MON_TABLE	32	Uint16
dft_mon_uow	SQLF_KTN_DFT_MON_UOW	30	Uint16
dftdbpath	SQLF_KTN_DFTDBPATH	27	char(215)
diaglevel	SQLF_KTN_DIAGLEVEL	64	Uint16
diagpath	SQLF_KTN_DIAGPATH	65	char(215)
dir_cache	SQLF_KTN_DIR_CACHE	40	Uint16
dir_obj_name	SQLF_KTN_DIR_OBJ_NAME	75	char(255)
dir_path_name	SQLF_KTN_DIR_PATH_NAME	74	char(255)
dir_type	SQLF_KTN_DIR_TYPE	73	Uint16
discover	SQLF_KTN_DISCOVER	304	Uint16
discover_comm	SQLF_KTN_DISCOVER_COMM	305	char(35)
discover_inst	SQLF_KTN_DISCOVER_INST	308	Uint16
dos_rqrioblk	SQLF_KTN_DOS_RQRIOBLK	72	Uint16
drda_heap_sz	SQLF_KTN_DRDA_HEAP_SZ	41	Uint16
fcm_num_anchors	SQLF_KTN_FCM_NUM_ANCHORS	506	Sint32
fcm_num_buffers	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32
fcm_num_connect	SQLF_KTN_FCM_NUM_CONNECT	505	Sint32
fcm_num_rqb	SQLF_KTN_FCM_NUM_RQB	504	Uint32
fileserver	SQLF_KTN_FILESERVER	47	char(48)
indexrec	SQLF_KTN_INDEXREC	20	Uint16
intra_parallel	SQLF_KTN_INTRA_PARALLEL	306	Sint16
ipx_socket	SQLF_KTN_IPX_SOCKET	71	char(4)
java_heap_sz	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32
jdk11_path	SQLF_KTN_JDK11_PATH	311	char(255)
keepdari	SQLF_KTN_KEEPDARI	81	Uint16
max_connretries	SQLF_KTN_MAX_CONNRETRIES	509	Uint16
max_coordagents	SQLF_KTN_MAX_COORDAGENTS	501	Sint32
max_querydegree	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32
max_time_diff	SQLF_KTN_MAX_TIME_DIFF	510	Uint16
maxagents	SQLF_KTN_MAXAGENTS	12	Uint32
maxcagents	SQLF_KTN_MAXCAGENTS	13	Sint32
maxdari	SQLF_KTN_MAXDARI	80	Sint32
maxtotfilop	SQLF_KTN_MAXTOTFILOP	45	Uint16
min_priv_mem	SQLF_KTN_MIN_PRIV_MEM	43	Uint32
mon_heap_sz	SQLF_KTN_MON_HEAP_SZ	79	Uint16

Table 44 (Page 3 of 3). Updateable Database Manager Configuration Parameters

Parameter Name	Token	Token Value	Data Type
nname	SQLF_KTN_NNAME	7	char(8)
num_initagents	SQLF_KTN_NUM_INITAGENTS	500	Uint32
num_poolagents	SQLF_KTN_NUM_POOLAGENTS	502	Sint32
numdb	SQLF_KTN_NUMDB	6	Uint16
objectname	SQLF_KTN_OBJECTNAME	48	char(48)
priv_mem_thresh	SQLF_KTN_PRIV_MEM_THRESH	44	Sint32
query_heap_sz	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32
restbufsz	SQLF_KTN_RESTBUFSZ	19	Uint32
resync_interval	SQLF_KTN_RESYNC_INTERVAL	68	Uint16
route_obj_name	SQLF_KTN_ROUTE_OBJ_NAME	76	char(255)
rqrioblk	SQLF_KTN_RQRIOBLK	1	Uint16
sheapthres	SQLF_KTN_SHEAPTHRES	21	Uint32
spm_name	SQLF_KTN_SPM_NAME	92	char(8)
spm_log_file_sz	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32
spm_max_resync	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32
ss_logon	SQLF_KTN_SS_LOGON	309	Uint16
start_stop_time	SQLF_KTN_START_STOP_TIME	511	Uint16
svcename	SQLF_KTN_SVCENAME	24	char(14)
sysadm_group	SQLF_KTN_SYSADM_GROUP	39	char(16)
sysctrl_group	SQLF_KTN_SYSCTRL_GROUP	63	char(16)
sysmaint_group	SQLF_KTN_SYSMAINT_GROUP	62	char(16)
tm_database	SQLF_KTN_TM_DATABASE	67	char(8)
tp_mon_name	SQLF_KTN_TP_MON_NAME	66	char(19)
tpname	SQLF_KTN_TPNAME	25	char(64)
trust_allclnts	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16
trust_clntauth	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16
udf_mem_sz	SQLF_KTN_UDF_MEM_SZ	69	Uint16

[a] 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
```

Table 45. Non-updateable Database Manager Configuration Parameters

Parameter Name	Token	Token Value	Data Type
nodetype	SQLF_KTN_NODETYPE	100	Uint16
release	SQLF_KTN_RELEASE	101	Uint16

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLFUPD */
/* ... */
SQL_STRUCTURE sqlfupd
{
  unsigned short  token;
  char            *ptrvalue;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-FUPD.
    05 SQL-TOKEN              PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
    05 SQL-VALUE-PTR          USAGE IS POINTER.
*
```

SQLM-COLLECTED

This structure is used to return information after a call to the Database System Monitor APIs.

Table 46. Fields in the SQLM-COLLECTED Structure

Field Name	Data Type	Description
SIZE	UNSIGNED LONG	The size of the structure.
DB2	UNSIGNED LONG	Obsolete.
DATABASES	UNSIGNED LONG	Obsolete.
TABLE_DATABASES	UNSIGNED LONG	Obsolete.
LOCK_DATABASES	UNSIGNED LONG	Obsolete.
APPLICATIONS	UNSIGNED LONG	Obsolete.
APPLINFOS	UNSIGNED LONG	Obsolete.
DCS_APPLINFOS	UNSIGNED LONG	Obsolete.
SERVER_DB2_TYPE	UNSIGNED LONG	The database manager server type (defined in `sqlutil.h`).
TIME_STAMP	TIMESTAMP	Time that the snapshot was taken.
GROUP_STATES	OBJECT SQLM_ RECORDING_ GROUP	Current state of the monitor switch.
SERVER_PRDID	CHAR(20)	Product name and version number of the database manager on the server.
SERVER_NNAME	CHAR(20)	Configuration node name of the server.
SERVER_ INSTANCE_NAME	CHAR(20)	Instance name of the database manager.
RESERVED	CHAR(22)	Reserved for future use.
NODE_NUMBER	UNSIGNED SHORT	Number of the node sending data.
TIME_ZONE_DISP	LONG	The difference (in seconds) between GMT and local time.
NUM_TOP_LEVEL_ STRUCTS	UNSIGNED LONG	The total number of high-level structures returned in the snapshot output buffer. A high-level structure can be composed of several lower-level data structures. This counter replaces the individual counters (such as *table_databases*) for each high-level structure, which are now obsolete.
TABLESPACE_ DATABASES	UNSIGNED LONG	Obsolete.
SERVER_VERSION	UNSIGNED LONG	The version of the server returning the data.

For information about programming the database monitor, see the *System Monitor Guide and Reference*.

SQLM-COLLECTED

Language Syntax

C Structure

```
/* File: sqlmon.h */
/* Structure: SQLM-COLLECTED */
/* ... */
typedef struct sqlm_collected
{
  unsigned long  size;
  unsigned long  db2;
  unsigned long  databases;
  unsigned long  table_databases;
  unsigned long  lock_databases;
  unsigned long  applications;
  unsigned long  applinfos;
  unsigned long  dcs_applinfos;
  unsigned long  server_db2_type;
  sqlm_timestamp time_stamp;
  sqlm_recording_group group_states[SQLM_NUM_GROUPS];
  _SQLOLDCHAR    server_prdid[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_nname[SQLM_IDENT_SZ];
  _SQLOLDCHAR    server_instance_name[SQLM_IDENT_SZ];
  _SQLOLDCHAR    reserved[22];
  unsigned short node_number;
  long           time_zone_disp;
  unsigned long  num_top_level_structs;
  unsigned long  tablespace_databases;
  unsigned long  server_version;
}sqlm_collected;
/* ... */
```

COBOL Structure

```
* File: sqlmonct.cbl
01 SQLM-COLLECTED.
    05 SQLM-SIZE               PIC 9(9) COMP-5.
    05 DB2                     PIC 9(9) COMP-5.
    05 DATABASES               PIC 9(9) COMP-5.
    05 TABLE-DATABASES         PIC 9(9) COMP-5.
    05 LOCK-DATABASES          PIC 9(9) COMP-5.
    05 APPLICATIONS            PIC 9(9) COMP-5.
    05 APPLINFOS               PIC 9(9) COMP-5.
    05 DCS-APPLINFOS           PIC 9(9) COMP-5.
    05 SERVER-DB2-TYPE         PIC 9(9) COMP-5.
    05 TIME-STAMP.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
    05 GROUP-STATES OCCURS 6.
        10 INPUT-STATE          PIC 9(9) COMP-5.
        10 OUTPUT-STATE         PIC 9(9) COMP-5.
        10 START-TIME.
    05 SERVER-PRDID            PIC X(20).
    05 SERVER-NNAME            PIC X(20).
    05 SERVER-INSTANCE-NAME    PIC X(20).
    05 RESERVED                PIC X(32).
    05 TABLESPACE-DATABASES    PIC 9(9) COMP-5.
    05 SERVER-VERSION          PIC 9(9) COMP-5.
*
```

SQLM-RECORDING-GROUP

This structure is used to return information after a call to the Database System Monitor APIs.

Table 47. Fields in the SQLM-RECORDING-GROUP Structure

Field Name	Data Type	Description
INPUT_STATE	INTEGER	Required state for the specific monitor group.
OUTPUT_STATE	INTEGER	Returned information on the state of the specific monitor switch.
START_TIME	Structure	Time stamp when the monitoring group switch was turned on.

Table 48. Fields in the SQLM-TIMESTAMP Structure

Field Name	Data Type	Description
SECONDS	INTEGER	The date and time, expressed as the number of seconds since January 1, 1970 (GMT).
MICROSEC	INTEGER	The number of elapsed microseconds in the current second.

For both *input_state* and *output_state*, a particular monitor switch is identified by its index in the array passed to "sqlmon - Get/Update Monitor Switches" on page 212. The constants that map the indexes to the switches are called `SQLM_`*XXXX*`_SW`, where *XXXX* is the name of the monitor group. These constants are defined in `sqlmon.h`.

For information about programming the database monitor, see the *System Monitor Guide and Reference*.

Language Syntax

C Structure

```
/* File: sqlmon.h */
/* Structure: SQLM-RECORDING-GROUP */
/* ... */
typedef struct sqlm_recording_group
{
  unsigned long  input_state;
  unsigned long  output_state;
  sqlm_timestamp start_time;
}sqlm_recording_group;
/* ... */
/* File: sqlmon.h */
/* Structure: SQLM-TIMESTAMP */
/* ... */
typedef struct sqlm_timestamp
{
  unsigned long seconds;
  unsigned long microsec;
}sqlm_timestamp;
/* ... */
```

COBOL Structure

```
* File: sqlmonct.cbl
01 SQLM-RECORDING-GROUP OCCURS 6 TIMES.
    05 INPUT-STATE             PIC 9(9) COMP-5.
    05 OUTPUT-STATE            PIC 9(9) COMP-5.
    05 START-TIME.
        10 SECONDS              PIC 9(9) COMP-5.
        10 MICROSEC             PIC 9(9) COMP-5.
*
```

```
* File: sqlmonct.cbl
01 SQLM-TIMESTAMP.
    05 SECONDS                 PIC 9(9) COMP-5.
    05 MICROSEC                PIC 9(9) COMP-5.
*
```

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 49. Fields in the SQLMA Structure

Field Name	Data Type	Description
OBJ_NUM	INTEGER	Number of objects to be monitored.
OBJ_VAR	Array	An array of *sqlm_obj_struct* structures containing descriptions of objects to be monitored. The length of the array is determined by *OBJ_NUM*.

Table 50. Fields in the SQLM-OBJ-STRUCT Structure

Field Name	Data Type	Description
AGENT_ID	INTEGER	The application handle of the application to be monitored. Specified only if *OBJ_TYPE* requires an *agent_id* (application handle).
OBJ_TYPE	INTEGER	The type of object to be monitored.
OBJECT	CHAR(36)	The name of the object to be monitored. Specified only if *OBJ_TYPE* requires a name, such as *appl_id*, or a database alias.

Valid values for *OBJ_TYPE* (defined in `sqlmon`) are:

SQLMA_DB2
DB2 related information

SQLMA_DBASE
Database related information

SQLMA_APPL
Application information organized by the application ID

SQLMA_AGENT_ID
Application information organized by the agent ID

SQLMA_DBASE_TABLES
Table information for a database

SQLMA_DBASE_APPLS
Application information for a database

SQLMA_DBASE_APPLINFO
Summary application information for a database

SQLMA_DBASE_LOCKS
Locking information for a database

SQLMA_DBASE_ALL
Database information for all active databases in the database manager

SQLMA_APPL_ALL
Application information for all active applications in the database manager

SQLMA_APPLINFO_ALL
    Summary application information for all active applications in the database manager

SQLMA_DCS_APPLINFO_ALL
    Database Connection Services application information summary for all active applications in the database manager.

For information about programming the database monitor, see the *System Monitor Guide and Reference*.

Language Syntax

C Structure

```
/* File: sqlmon.h */
/* Structure: SQLMA */
/* ... */
typedef struct sqlma
{
  unsigned long obj_num;
  sqlm_obj_struct obj_var[1];
}sqlma;
/* ... */
```

```
/* File: sqlmon.h */
/* Structure: SQLM-OBJ-STRUCT */
/* ... */
typedef struct sqlm_obj_struct
{
  unsigned long agent_id;
  unsigned long obj_type;
  _SQLOLDCHAR   object[SQLM_OBJECT_SZ];
}sqlm_obj_struct;
/* ... */
```

COBOL Structure

```
* File: sqlmonct.cbl
01 SQLMA.
    05 OBJ-NUM                PIC 9(9) COMP-5.
    05 OBJ-VAR OCCURS 0 TO 100 TIMES DEPENDING ON OBJ-NUM.
        10 AGENT-ID            PIC 9(9) COMP-5.
        10 OBJ-TYPE            PIC 9(9) COMP-5.
        10 OBJECT              PIC X(36).
*
```

SQLOPT

This structure is used to pass bind options to "sqlabndx - Bind" on page 10, and precompile options to "sqlaprep - Precompile Program" on page 18.

Table 51. 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 52. 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 53. Fields in the SQLOPTIONS Structure

Field Name	Data Type	Description
TYPE	INTEGER	Bind/precompile option type.
VAL	INTEGER	Bind/precompile option value.
Note: The *TYPE* and *VAL* fields are repeated for each bind/precompile option specified.		

For more information about valid values for *TYPE* and *VAL*, see "sqlabndx - Bind" on page 10 and "sqlaprep - Precompile Program" on page 18.

Language Syntax

C Structure

```
/* File: sql.h */
/* Structure: SQLOPT */
/* ... */
SQL_STRUCTURE sqloptheader
{
  unsigned long allocated;
  unsigned long used;
};
/* ... */
/* File: sql.h */
/* Structure: SQLOPTHEADER */
/* ... */
SQL_STRUCTURE sqloptheader
{
  unsigned long allocated;
  unsigned long used;
};
/* ... */
```

```
/* File: sql.h */
/* Structure: SQLOPTIONS */
/* ... */
SQL_STRUCTURE sqloptions
{
  unsigned long type;
  unsigned long 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, used by “sqlurlog - Asynchronous Read Log” on page 297, contains the definition of the log sequence number. A log sequence number (lsn) represents a relative byte address within the database log. All log records are identified by this number. It represents the log record’s byte offset from the beginning of the database log.

Table 54. Fields in the SQLU-LSN Union

Field Name	Data Type	Description
lsnChar	Array of UNSIGNED CHAR	Specifies the 6-member character array log sequence number.
lsnWord	Array of UNSIGNED SHORT	Specifies the 3-member short array log sequence number.

Language Syntax

C Structure

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

SQLU-MEDIA-LIST

This structure is used to:

- Hold a list of *target* media for the backup image (see “sqlubkp - Backup Database” on page 230)
- Hold a list of *source* media for the backup image (see “sqlurst - Restore Database” on page 309)
- Pass information to “sqluload - Load” on page 282.

Table 55. Fields in the SQLU-MEDIA-LIST Structure

Field Name	Data Type	Description
MEDIA_TYPE	CHAR(1)	A character indicating media type.
SESSIONS	INTEGER	Indicates the number of elements in the array pointed to by the *target* field of this structure.
TARGET	Union	This field is a pointer to one of three types of structures. The type of structure pointed to is determined by the value of the *media_type* field. For more information on what to provide in this field, see the appropriate API.

Table 56. Fields in the SQLU-MEDIA-LIST-TARGETS Structure

Field Name	Data Type	Description
MEDIA	Pointer	A pointer to an *sqlu_media_entry* structure.
VENDOR	Pointer	A pointer to an *sqlu_vendor* structure.
LOCATION	Pointer	A pointer to an *sqlu_location_entry* structure.

Table 57. Fields in the SQLU-MEDIA-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the *media_entry* field. For languages other than C.
MEDIA_ENTRY	CHAR(215)	Path for a backup image used by the backup and restore utilities.

Table 58. Fields in the SQLU-VENDOR Structure

Field Name	Data Type	Description
RESERVE_LEN1	INTEGER	Length of the *shr_lib* field. For languages other than C.
SHR_LIB	CHAR(255)	Name of a shared library supplied by vendors for storing or retrieving data.
RESERVE_LEN2	INTEGER	Length of the *filename* field. For languages other than C.
FILENAME	CHAR(255)	File name to identify the load input source when using a shared library.

Table 59. Fields in the SQLU-LOCATION-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the *location_entry* field. For languages other than C.
LOCATION_ENTRY	CHAR(256)	Name of input data files for the load utility.

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 *pDataFileList* parameter.

SQLU_ADSM_MEDIA
ADSM

SQLU_OTHER_MEDIA
Vendor library

SQLU_USER_EXIT
User exit (OS/2 only)

SQLU_PIPE_MEDIA
Named pipe (for vendor APIs only)

SQLU_DISK_MEDIA
Disk (for vendor APIs only)

SQLU_DISKETTE_MEDIA
Diskette (for vendor APIs only)

SQLU_TAPE_MEDIA
Tape (for vendor APIs only).

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_media_list
{
  char            media_type;
  char            filler[3];
  long            sessions;
  union sqlu_media_list_targets target;
} sqlu_media_list;
/* ... */
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-LIST-TARGETS */
/* ... */
union sqlu_media_list_targets
{
  struct sqlu_media_entry        *media;
  struct sqlu_vendor             *vendor;
  struct sqlu_location_entry     *location;
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-MEDIA-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_media_entry
{
  unsigned long   reserve_len;
  char            media_entry[SQLU_DB_DIR_LEN+1];
} sqlu_media_entry;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-VENDOR */
/* ... */
typedef SQL_STRUCTURE sqlu_vendor
{
  unsigned long   reserve_len1;
  char            shr_lib[SQLU_SHR_LIB_LEN+1];
  unsigned long   reserve_len2;
  char            filename[SQLU_SHR_LIB_LEN+1];
} sqlu_vendor;
/* ... */
/* File: sqlutil.h */
/* Structure: SQLU-LOCATION-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_location_entry
{
  unsigned long   reserve_len;
  char            location_entry[SQLU_MEDIA_LOCATION_LEN+1];
} sqlu_location_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 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.
*
```

SQLU-RLOG-INFO

This structure contains information regarding calls to “sqlurlog - Asynchronous Read Log” on page 297. The read log information structure contains information on the status of the call and the database log.

Table 60. Fields in the SQLU-RLOG-INFO Structure

Field Name	Data Type	Description
initialLSN	SQLU_LSN	Specifies the lsn value of the first log record written to the database after the first *connect* is issued. For more information on the *SQLU_LSN* structure, see “SQLU-LSN” on page 416.
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 byte read.
curActiveLSN	SQLU_LSN	Specifies the lsn value of the current active log.
logRecsWritten	UNSIGNED LONG	Specifies the number of log records written to the buffer.
logBytesWritten	UNSIGNED LONG	Specifies the number of bytes written to the buffer.

Language Syntax

C Structure

```
typedef SQL_STRUCTURE SQLU_RLOG_INFO
{
SQLU_LSN       initialLSN ;
SQLU_LSN       firstReadLSN ;
SQLU_LSN       lastReadLSN ;
SQLU_LSN       curActiveLSN ;
unsigned long  logRecsWritten ;
unsigned long  logBytesWritten ;
} SQLU_RLOG_INFO;
```

SQLU-TABLESPACE-BKRST-LIST

This structure is used to provide a list of table space names.

Table 61. Fields in the SQLU-TABLESPACE-BKRST-LIST Structure

Field Name	Data Type	Description
NUM_ENTRY	INTEGER	Number of entries in the list pointed to by the *tablespace* field.
TABLESPACE	Pointer	A pointer to an *sqlu_tablespace_entry* structure.

Table 62. Fields in the SQLU-TABLESPACE-ENTRY Structure

Field Name	Data Type	Description
RESERVE_LEN	INTEGER	Length of the character string provided in the *tablespace_entry* field. For languages other than C.
TABLESPACE_ENTRY	CHAR(19)	Table space name.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLU-TABLESPACE-BKRST-LIST */
/* ... */
typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
  long            num_entry;
  struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLU-TABLESPACE-ENTRY */
/* ... */
typedef SQL_STRUCTURE sqlu_tablespace_entry
{
  unsigned long   reserve_len;
  char            tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
  char            filler[1];
} sqlu_tablespace_entry;
/* ... */
```

SQLU-TABLESPACE-BKRST-LIST

COBOL Structure

```
* File: sqlutil.cbl
01 SQLU-TABLESPACE-BKRST-LIST.
    05 SQL-NUM-ENTRY          PIC S9(9) COMP-5.
    05 SQL-TABLESPACE         USAGE IS POINTER.
*
```

```
* File: sqlutil.cbl
01 SQLU-TABLESPACE-ENTRY.
    05 SQL-TBSP-LEN           PIC 9(9) COMP-5.
    05 SQL-TABLESPACE-ENTRY   PIC X(18).
    05 FILLER                 PIC X.
    05 SQL-FILLER             PIC X(1).
*
```

SQLUEXPT-OUT

This structure is used to pass information from “sqluexpr - Export” on page 241.

Table 63. Fields in the SQLUEXPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of the structure.
ROWSEXPORTED	INTEGER	Number of records exported from the database into the target file.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-UEXPT-OUT */
/* ... */
SQL_STRUCTURE sqluexpt_out
{
  unsigned long   sizeOfStruct;
  unsigned long   rowsExported;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UEXPT-OUT.
    05 SQL-SIZE-OF-UEXPT-OUT  PIC 9(9) COMP-5 VALUE 8.
    05 SQL-ROWSEXPORTED       PIC 9(9) COMP-5 VALUE 0.
*
```

SQLUHINFO

This structure is used to return information after a call to “sqluhgne - Get Next Recovery History File Entry” on page 256.

Table 64 (Page 1 of 2). Fields in the SQLUHINFO Structure

Field Name	Data Type	Description
SQLUHINFOID	CHAR(8)	A structure identifier and "eye-catcher" for storage dumps. It is a string of eight bytes that must be initialized with the string `"SQLUHINF"`. No symbolic definition for this string exists.
SQLUHINFOBC	INTEGER	Size of this structure in bytes. Use the `SQLUHINFOSIZE` macro (defined in `sqlutil`) to set this field.
SQLN	SMALLINT	Number of table space elements.
SQLD	SMALLINT	Number of used table space elements.
OPERATION	CHAR(1)	Type of operation performed: `B` for backup, `R` for restore, `U` for unload, and `L` for load.
OBJECT	CHAR(1)	Granularity of the operation: `D` for full database, `P` for table space, and `T` for table.
OBJECT_PART	CHAR(17)	The first 14 characters are a time stamp with format *yyyymmddhhnnss*, indicating when the operation was done. The next 3 characters are a sequence number. Each backup and unload operation can result in multiple entries in this file when the backup image or unload 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 or unload. The time stamp, combined with the sequence number, must be unique.
OPTYPE	CHAR(1)	Operation type. Additional qualification of the operation. For a full database or table space level backup: `F` indicates an offline backup, and `N` indicates an online backup. For a load: `R` indicates replace, `A` indicates append, and `C` indicates copy. Any other operation will leave this field blank.
DEVICE_TYPE	CHAR(1)	Device type. This field determines how the *LOCATION* field is interpreted: `D` indicates disk, `K` indicates diskette, `T` indicates tape, `A` indicates ADSM, `U` indicates user exit, and `O` indicates other (for other vendor device support).
FIRST_LOG	CHAR(12)	The earliest log file ID (ranging from S0000000 to S9999999): • Required to apply roll forward recovery for an online backup • Required to apply roll forward recovery for an offline backup • Applied after restoring a full database or table space level backup that was current when the unload/load started.

Table 64 (Page 2 of 2). Fields in the SQLUHINFO Structure

Field Name	**Data Type**	**Description**
LAST_LOG	CHAR(12)	The latest log file ID (ranging from S0000000 to S9999999): • Required to apply roll forward recovery for an online backup • Required to apply roll forward recovery to the current point in time for an offline backup • Applied after restoring a full database or table space level backup that was current when the unload/load finished (will be the same as *FIRST_LOG* if roll forward recovery is not applied).
BACKUP_ID	CHAR(14)	A time stamp with format *yyyymmddhhnnss* that references one or more file lines (depending on sequence number) representing backup operations. For a full database restore, this references the full database backup that was restored. For a table space restore, this references the table space backup, or full database backup used to restore the specified table spaces. This field is otherwise left blank.
TABLE_CREATOR	CHAR(8)	Table creator. Blank except for unload and load operations.
TABLE_NAME	CHAR(18)	Table name. Blank except for unload and load operations.
NUM_OF_ TABLESPACES	CHAR(5)	Number of table spaces involved in the backup or restore. Each table space backup contains one or more table spaces. Each table space restore 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.
LOCATION	CHAR(255)	For backups, and copies for loads and unloads, this field indicates where the data has been saved. For operations that require multiple entries in the file, the sequence number defined by *OBJECT_PART* identifies which part of the backup or unload is found in the specified location. For restores and loads, the location always identifies where the first part of the data restored or loaded (corresponding to sequence '001' for multi-part backups and unloads) has been saved. The data in *LOCATION* is interpreted differently, depending on *DEVICE_TYPE*: • For disk or diskette (*D* or *K*), a fully qualified file name • For tape (*T*), a volume label • For ADSM (*A*), the server name • For user exit or other (*U* or *O*), free form text.
COMMENT	CHAR(30)	Free form text comment.
TABLESPACE	Array	An array of *SQLN sqluhtsp* structures.

Table 65. Fields in the SQLUHTSP Structure

Field Name	Data Type	Description
TABLESPACE_NAME	CHAR(18)	A string containing the name of a table space.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUHINFO */
/* ... */
SQL_STRUCTURE sqluhinfo
{
  char      sqluhinfoid[8];
  long      sqluhinfobc;
  short     sqln;
  short     sqld;
  char      operation[SQLUH_OP_SZ+1];
  char      object[SQLUH_OBJ_SZ+1];
  char      object_part[SQLUH_OBJPART_SZ+1];
  char      optype[SQLUH_OPTYPE_SZ+1];
  char      device_type[SQLUH_DEVTYPE_SZ+1];
  char      first_log[SQLUH_FIRSTLOG_SZ+1];
  char      last_log[SQLUH_LASTLOG_SZ+1];
  char      backup_id[SQLUH_BACKID_SZ+1];
  char      table_creator[SQLUH_TCREATE_SZ+1];
  char      table_name[SQLUH_TNAME_SZ+1];
  char      num_of_tablespaces[SQLUH_NUMTABLESPACE_SZ+1];
  char      location[SQLUH_LOC_SZ+1];
  char      comment[SQLUH_COMMENT_SZ+1];
  struct sqluhtsp tablespace[1];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLUHTSP */
/* ... */
SQL_STRUCTURE sqluhtsp
{
  char            tablespace_name[SQLUH_TABLESPACENAME_SZ+1];
  char            filler;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLUHINFO.
    05 SQLUHINFOID             PIC X(8).
    05 SQLUHINFOBC             PIC S9(9) COMP-5.
    05 SQLH-SQLN               PIC S9(4) COMP-5.
    05 SQLH-SQLD               PIC S9(4) COMP-5.
    05 SQL-OPERATION           PIC X(1).
    05 FILLER                  PIC X.
    05 SQL-OBJECT              PIC X(1).
    05 FILLER                  PIC X.
    05 SQL-OBJECT-PART         PIC X(17).
    05 FILLER                  PIC X.
    05 SQL-OPTYPE              PIC X(1).
    05 FILLER                  PIC X.
    05 SQL-DEVICE-TYPE         PIC X(1).
    05 FILLER                  PIC X.
    05 SQL-FIRST-LOG           PIC X(12).
    05 FILLER                  PIC X.
    05 SQL-LAST-LOG            PIC X(12).
    05 FILLER                  PIC X.
    05 SQL-BACKUP-ID           PIC X(14).
    05 FILLER                  PIC X.
    05 SQL-TABLE-CREATOR       PIC X(8).
    05 FILLER                  PIC X.
    05 SQL-TABLE-NAME          PIC X(18).
    05 FILLER                  PIC X.
    05 SQL-NUM-OF-TABLESPACES PIC X(5).
    05 FILLER                  PIC X.
    05 SQL-LOCATION            PIC X(255).
    05 FILLER                  PIC X.
    05 SQL-COMMENT             PIC X(30).
    05 FILLER                  PIC X.
    05 SQL-TABLESPACE OCCURS 1 TIMES.
        10 SQL-TABLESPACE-NAME PIC X(18).
        10 FILLER              PIC X.
        10 SQL-FILLER          PIC X.
*
```

```
* File: sqlutil.cbl
01 SQLUHTSP.
    05 SQL-TABLESPACE-NAME     PIC X(18).
    05 FILLER                  PIC X.
    05 SQL-FILLER              PIC X.
*
```

SQLUIMPT-IN

This structure is used to pass information to "sqluimpr - Import" on page 271.

Table 66. Fields in the SQLUIMPT-IN Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
COMMITCNT	INTEGER	The number of records to import before committing them to the database. A COMMIT is performed whenever *commitcnt* records are imported.
RESTARTCNT	INTEGER	The number of records to skip before starting to insert/update records. This parameter should be used if a previous attempt to import records fails after some records have been committed to the database. The parameter's value represents a starting point for the next import.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-IN */
/* ... */
SQL_STRUCTURE sqluimpt_in
{
  unsigned long   sizeOfStruct;
  unsigned long   commitcnt;
  unsigned long   restartcnt;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-IN.
    05 SQL-SIZE-OF-UIMPT-IN   PIC 9(9) COMP-5 VALUE 12.
    05 SQL-COMMITCNT          PIC 9(9) COMP-5 VALUE 0.
    05 SQL-RESTARTCNT         PIC 9(9) COMP-5 VALUE 0.
*
```

SQLUIMPT-OUT

This structure is used to pass information from “sqluimpr - Import” on page 271.

Table 67. Fields in the SQLUIMPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
ROWSREAD	INTEGER	Number of records read from the file during import.
ROWSSKIPPED	INTEGER	Number of records skipped before inserting or updating begins.
ROWSINSERTED	INTEGER	Number of rows inserted into the target table.
ROWSUPDATED	INTEGER	Number of rows in the target table updated with information from the imported records (records with the same key already exist in the table).
ROWSREJECTED	INTEGER	Number of records that could not be imported.
ROWSCOMMITTED	INTEGER	Number of records imported successfully and committed to the database.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-OUT */
/* ... */
SQL_STRUCTURE sqluimpt_out
{
  unsigned long   sizeOfStruct;
  unsigned long   rowsRead;
  unsigned long   rowsSkipped;
  unsigned long   rowsInserted;
  unsigned long   rowsUpdated;
  unsigned long   rowsRejected;
  unsigned long   rowsCommitted;
};
/* ... */
```

SQLUIMPT-OUT

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-OUT.
    05 SQL-SIZE-OF-UIMPT-OUT  PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWSREAD           PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSSKIPPED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSINSERTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSUPDATED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSREJECTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSCOMMITTED      PIC 9(9) COMP-5 VALUE 0.
*
```

SQLULOAD-IN

This structure is used to input information during a call to "sqluload - Load" on page 282.

Table 68 (Page 1 of 2). Fields in the SQLULOAD-IN Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	UNSIGNED LONG	Size of this structure in bytes.
SAVECNT	UNSIGNED LONG	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 will be monitored using "sqluqry - Load Query" on page 291. If the value of *savecnt* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance. The default value is `0`, meaning that no consistency points will be established, unless necessary.
RESTARTCNT	UNSIGNED LONG	The number of records to skip before starting to load records. This parameter should be used if a previous attempt to load records fails after some records have been committed to the database. The parameter's value represents a starting point for the next load.
ROWCNT	UNSIGNED LONG	The number of physical records to be loaded. Allows a user to load only the first *rowcnt* rows in a file.
WARNINGCNT	UNSIGNED LONG	Stops the load after *warningcnt* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If *warningcnt* is `0`, or this option is not specified, the load will continue regardless of the number of warnings issued. If the load is stopped because the threshold of warnings was encountered, another load can be started in RESTART mode by specifying the *restartcnt* option. Alternatively, another load can be initiated in REPLACE mode, starting at the beginning of the input file.
DATA_BUFFER_SIZE	UNSIGNED LONG	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 minimum required resource is used, and no warning is returned. This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter. If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

Table 68 (Page 2 of 2). Fields in the SQLULOAD-IN Structure

Field Name	Data Type	Description
SORT_BUFFER_SIZE	UNSIGNED LONG	The number of 4KB pages of memory that are to be used for sorting the index keys during a load operation. **Note:** Sort buffer size has a very large impact on sort performance. Therefore, for very large tables (for example, tables in excess of 100M), this buffer should be set as large as possible. If a value is not specified, the utility uses the larger of: • 2MB for OS/2 or Windows NT, or 6MB for all other platforms • The minimum size allowed by the sort algorithm • 15% of the free space remaining in the utility heap. If a value greater than zero, but less than the required minimum is specified, the minimum value for that load is returned.
HOLD_QUIESCE	UNSIGNED SHORT	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.
RESTARTPHASE	CHAR(1)	Phase at which to restart the load operation. See below for values.
STATSOPT	CHAR(1)	Granularity of statistics to collect. See below for values.
CPU_PARALLELISM	UNSIGNED SHORT	The number of processes or threads that the load utility will spawn for parsing, converting and formatting records when building table objects. This parameter is designed to exploit SMP 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.
DISK_PARALLELISM	UNSIGNED SHORT	The number of processes or threads that the load utility will spawn for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.
NON_RECOVERABLE	UNSIGNED SHORT	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 rollforward 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. Set to `SQLU_RECOVERABLE_LOAD` if the load transaction is to be marked as recoverable.

Valid values for *RESTARTPHASE* (defined in `sqlutil`) are:

SQLU_LOAD_PHASE
Restart at load phase.

SQLU_BUILD_PHASE
Restart at build phase.

SQLU_DELETE_PHASE
Restart at delete phase.

Valid values for *STATSOPT* (defined in `sqlutil`) are:

SQLU_STATS_NONE
SQL_STATS_EXTTABLE_ONLY
SQL_STATS_EXTTABLE_INDEX
SQL_STATS_INDEX
SQL_STATS_TABLE
SQL_STATS_EXTINDEX_ONLY
SQL_STATS_EXTINDEX_TABLE
SQL_STATS_ALL
SQL_STATS_BOTH

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLULOAD-IN */
/* ... */
SQL_STRUCTURE sqluload_in
{
  unsigned long   sizeOfStruct;
  unsigned long   savecnt;
  unsigned long   restartcnt;
  unsigned long   rowcnt;
  unsigned long   warningcnt;
  unsigned long   data_buffer_size;
  unsigned long   sort_buffer_size;
  unsigned short  hold_quiesce;
  char            restartphase;
  char            statsopt;
  unsigned short  cpu_parallelism;
  unsigned short  disk_parallelism;
  unsigned short  non_recoverable;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLULOAD-IN.
    05 SQL-SIZE-OF-STRUCT     PIC 9(9) COMP-5 VALUE 40.
    05 SQL-SAVECNT            PIC 9(9) COMP-5.
    05 SQL-RESTARTCOUNT       PIC 9(9) COMP-5.
    05 SQL-ROWCNT             PIC 9(9) COMP-5.
    05 SQL-WARNINGCNT         PIC 9(9) COMP-5.
    05 SQL-DATA-BUFFER-SIZE   PIC 9(9) COMP-5.
    05 SQL-SORT-BUFFER-SIZE   PIC 9(9) COMP-5.
    05 SQL-HOLD-QUIESCE       PIC 9(4) COMP-5.
    05 SQL-RESTARTPHASE       PIC X.
    05 SQL-STATSOPT           PIC X.
    05 SQL-CPU-PARALLELISM    PIC 9(4) COMP-5.
    05 SQL-DISK-PARALLELISM   PIC 9(4) COMP-5.
    05 SQL-NON-RECOVERABLE    PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
*
```

SQLULOAD-OUT

This structure is used to output information after a call to “sqluload - Load” on page 282.

Table 69. Fields in the SQLULOAD-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	UNSIGNED LONG	Size of this structure in bytes.
ROWSREAD	UNSIGNED LONG	Number of records read during the load.
ROWSSKIPPED	UNSIGNED LONG	Number of records skipped before the load begins.
ROWSLOADED	UNSIGNED LONG	Number of rows loaded into the target table.
ROWSREJECTED	UNSIGNED LONG	Number of records that could not be loaded.
ROWSDELETED	UNSIGNED LONG	Number of duplicate rows deleted.
ROWSCOMMITTED	UNSIGNED LONG	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.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLULOAD-OUT */
/* ... */
SQL_STRUCTURE sqluload_out
{
  unsigned long   sizeOfStruct;
  unsigned long   rowsRead;
  unsigned long   rowsSkipped;
  unsigned long   rowsLoaded;
  unsigned long   rowsRejected;
  unsigned long   rowsDeleted;
  unsigned long   rowsCommitted;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLULOAD-OUT.
    05 SQL-SIZE-OF-STRUCT     PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWS-READ          PIC 9(9) COMP-5.
    05 SQL-ROWS-SKIPPED       PIC 9(9) COMP-5.
    05 SQL-ROWS-LOADED        PIC 9(9) COMP-5.
    05 SQL-ROWS-REJECTED      PIC 9(9) COMP-5.
    05 SQL-ROWS-DELETED       PIC 9(9) COMP-5.
    05 SQL-ROWS-COMMITTED     PIC 9(9) COMP-5.
*
```

SQLUPI

This structure is used to store partitioning information, such as the partitioning map and the partitioning key of a table.

Table 70. Fields in the SQLUPI Structure

Field Name	Data Type	Description
PMAPLEN	INTEGER	The length of the partitioning map in bytes. For a single-node table, the value is `sizeof(SQL_PDB_NODE_TYPE)`. For a mult-inode table, the value is `SQL_PDB_MAP_SIZE * sizeof(SQL_PDB_NODE_TYPE)`.
PMAP	SQL_PDB_NODE_TYPE	The partitioning map.
SQLD	INTEGER	The number of used SQLPARTKEY elements; that is, the number of keyparts in a partitioning key.
SQLPARTKEY	Structure	The description of a partitioning column in a partitioning key. The maximum number of partitioning columns is SQL_MAX_NUM_PART_KEYS.

Table 71 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 71 (Page 1 of 2). 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 71 (Page 2 of 2). SQL Data Types and Lengths for the SQLUPI Structure

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

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUPI */
/* ... */
SQL_STRUCTURE sqlupi
{
  unsigned short  pmaplen;
  SQL_PDB_NODE_TYPE pmap[SQL_PDB_MAP_SIZE];
  unsigned short  sqld;
  struct sqlpartkey sqlpartkey[SQL_MAX_NUM_PART_KEYS];
};
/* ... */
```

```
/* File: sqlutil.h */
/* Structure: SQLPARTKEY */
/* ... */
SQL_STRUCTURE sqlpartkey
{
  unsigned short  sqltype;
  unsigned short  sqllen;
};
/* ... */
```

SQLXA-RECOVER

Used by the transaction APIs to return information about indoubt transactions (see Appendix B, "Transaction APIs" on page 447).

Table 72. Fields in the SQLXA-RECOVER Structure

Field Name	Data Type	Description
TIMESTAMP	INTEGER	Time stamp when the transaction entered the prepared (indoubt) state. This is the number of seconds the local time zone is displaced from Coordinated Universal Time.
XID	CHAR(140)	XA identifier assigned by the transaction manager to uniquely identify a global transaction.
DBALIAS	CHAR(16)	Alias of the database where the indoubt transaction is found.
APPLID	CHAR(30)	Application identifier assigned by the database manager for this transaction.
SEQUENCE_NO	CHAR(4)	The sequence number assigned by the database manager as an extension to the *APPLID*.
AUTH_ID	CHAR(8)	ID of the user who ran the transaction.
LOG_FULL	CHAR(1)	Indicates whether this transaction caused a log full condition.
CONNECTED	CHAR(1)	Indicates whether an application is connected.
INDOUBT_STATUS	CHAR(1)	Possible values are listed below.
RESERVED	CHAR(9)	The first byte is used to indicate the type of indoubt transaction: `0` indicates RM, and `1` indicates TM.

Possible values for *LOGFULL* (defined in `sqlxa`) are:

SQLXA_TRUE
True

SQLXA_FALSE
False.

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.

Possible values for *INDOUBT_STATUS* (defined in `sqlxa`) are:

SQLXA_TS_PREP
Prepared

SQLXA_TS_HCOM
Heuristically committed

SQLXA_TS_HROL
Heuristically rolled back

SQLXA_TS_END
Idle.

Language Syntax

C Structure

```
/* File: sqlxa.h */
/* Structure: SQLXA-RECOVER */
/* ... */
typedef struct sqlxa_recover_t
{
  unsigned long  timestamp;
  SQLXA_XID      xid;
  _SQLOLDCHAR    dbalias[SQLXA_DBNAME_SZ];
  _SQLOLDCHAR    applid[SQLXA_APPLID_SZ];
  _SQLOLDCHAR    sequence_no[SQLXA_SEQ_SZ];
  _SQLOLDCHAR    auth_id[SQLXA_USERID_SZ];
  char           log_full;
  char           connected;
  char           indoubt_status;
  char           originator;
  char           reserved[8];
} SQLXA_RECOVER;
/* ... */
```

SQLXA-XID

Used by the transaction APIs to identify XA transactions (see Appendix B, “Transaction APIs” on page 447).

Table 73. 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.		

Language Syntax

C Structure

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID */
/* ... */
typedef struct sqlxa_xid_t SQLXA_XID;
/* ... */
```

```
/* File: sqlxa.h */
/* Structure: SQLXA-XID-T */
/* ... */
struct sqlxa_xid_t
{
  long formatID;
  long gtrid_length;
  long bqual_length;
  char data[SQLXA_XIDDATASIZE];
};
/* ... */
```

Appendix A. Naming Conventions

This section provides information about the conventions that apply when naming database manager objects, such as databases and tables, and authentication IDs.

- Character strings that represent names of database manager objects can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.
- Unless otherwise noted, names can be entered in lowercase letters; however, the database manager processes them as if they were uppercase.

  The exception to this is character strings that represent names under the systems network architecture (SNA). Many values, such as logical unit names (partner_lu and local_lu), are case sensitive. The name must be entered exactly as it appears in the SNA definitions that correspond to those terms.
- A database name or database alias is a unique character string containing from one to eight letters, numbers, or keyboard characters from the set described above.

  Databases are cataloged in the system and local database directories by their aliases in one field, and their original name in another. For most functions, the database manager uses the name entered in the alias field of the database directories. (The exceptions are CHANGE DATABASE COMMENT and CREATE DATABASE, where a directory path must be specified.)
- The long identifier or alias for a database table or view, and the name of a column within a table or a view, are unique character strings 1 to 18 characters in length.

  A fully qualified table name consists of the *schema.tablename*. The schema is the unique user ID under which the table was created.
- Authentication IDs (both user IDs and group IDs) cannot exceed eight characters in length.

For more information about naming conventions, see the *Administration Guide*.

Appendix B. Transaction APIs

Databases can be used in a distributed transaction processing (DTP) environment; for information about this topic and heuristic operations, see the *Administration Guide*.

Heuristic APIs

A set of APIs is provided for tool writers to perform heuristic functions on indoubt transactions when the resource owner (such as the database administrator) cannot wait for the Transaction Manager (TM) to perform the *re-sync* action. This condition may occur if, for example, the communication line is broken, and an indoubt transaction is tying up needed resources. For the database manager, these resources include locks on tables and indexes, log space, and storage used by the transaction. Each indoubt transaction also decreases, by one, the maximum number of concurrent transactions that could be processed by the database manager.

The heuristic APIs have the capability to query, commit, and roll back indoubt transactions, and to cancel transactions that have been heuristically committed or rolled back, by removing the log records and releasing log pages.

Attention: The heuristic APIs should be used with caution and only as a last resort. The TM should drive the re-sync events. If the TM has an operator command to start the re-sync action, it should be used. If the user cannot wait for a TM-initiated re-sync, heuristic actions are necessary.

Although there is no set way to perform these actions, the following guidelines may be helpful:

- Use the **sqlxphqr** 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 - Forget Transaction Status” on page 449 unless a heuristically committed or rolled back transaction happens to cause a log full condition. The forget function releases the log space occupied by this indoubt transaction. If a transaction manager eventually performs a re-sync action for this indoubt transaction, the TM could make the wrong decision to commit or to roll back other RMs, because no record was found in this RM. In general, a missing record implies that the RM has rolled back.

sqlxhfrg - Forget Transaction Status

Permits the RM to erase knowledge of a heuristically completed transaction (that is, one that has been committed or rolled back heuristically).

Authorization

One of the following:

sysadm
dbadm

Required Connection

Database

API Include File

sqlxa.h

C API Syntax

```
/* File: sqlxa.h */
/* API: Forget Transaction Status */
/* ... */
extern int SQL_API_FN sqlxhfrg(
   SQLXA_XID            *pTransId,
   struct sqlca         *pSqlca
   );
/* ... */
```

API Parameters

pTransId
: Input. XA identifier of the transaction to be heuristically forgotten, or removed from the database log.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Usage Notes

Only transactions with a status of *heuristically committed* or *rolled back* can have the FORGET operation applied to them.

For information about the *SQLXA_XID* structure, see “SQLXA-XID” on page 443.

sqlxphcm - Commit an Indoubt Transaction

Commits an indoubt transaction (that is, a transaction that is prepared to be committed). If the operation succeeds, the transaction's state becomes *heuristically committed*.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
dbadm

Required Connection

Database

API Include File

sqlxa.h

C API Syntax

```
/* File: sqlxa.h */
/* API: Commit an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphcm(
   int                 exe_type,
   SQLXA_XID           *pTransId,
   struct sqlca        *pSqlca
   );
/* ... */
```

API Parameters

exe_type
: Input. If `EXE_THIS_NODE` is specified, the operation is executed only at this node.

pTransId
: Input. XA identifier of the transaction to be heuristically committed.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Usage Notes

Only transactions with a status of *prepared* can be committed. Once heuristically committed, the database manager remembers the state of the transaction until "sqlxhfrg - Forget Transaction Status" on page 449 is issued.

For information about the *SQLXA_XID* structure, see "SQLXA-XID" on page 443.

sqlxphqr - List Indoubt Transactions

Gets a list of all indoubt transactions for the currently connected database.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
dbadm

Required Connection

Database

API Include File

sqlxa.h

C API Syntax

```
/* File: sqlxa.h */
/* API: List Indoubt Transactions */
/* ... */
extern int SQL_API_FN sqlxphqr(
   int                  exe_type,
   SQLXA_RECOVER        **ppIndoubtData,
   long                 *pNumIndoubts,
   struct sqlca         *pSqlca
   );
/* ... */
```

API Parameters

exe_type
: Input. If `EXE_THIS_NODE` is specified, the operation is executed only at this node.

ppIndoubtData
: Output. Supply the address of a pointer to an *SQLXA_RECOVER* structure to hold the indoubt transactions. This API allocates sufficient space to hold the list of indoubt transactions, and returns a pointer to this space. The space is released only when the process terminates. Do not use "sqlefmem - Free Memory" on page 117 to free this memory, since it contains pointers to other dynamically allocated structures which will not be freed. For more information, see "SQLXA-RECOVER" on page 441.

pNumIndoubts
: Output. The API will return the number of indoubt transactions returned in *ppIndoubtData*.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqlxphrl - Roll Back an Indoubt Transaction

Rolls back an indoubt transaction (that is, a transaction that has been prepared). If the operation succeeds, the transaction's state becomes *heuristically rolled back*.

Scope

This API only affects the node on which it is issued.

Authorization

One of the following:

sysadm
dbadm

Required Connection

Database

API Include File

sqlxa.h

C API Syntax

```
/* File: sqlxa.h */
/* API: Roll Back an Indoubt Transaction */
/* ... */
extern int SQL_API_FN sqlxphrl(
   int                 exe_type,
   SQLXA_XID           *pTransId,
   struct sqlca        *pSqlca
   );
/* ... */
```

API Parameters

exe_type
: Input. If `EXE_THIS_NODE` is specified, the operation is executed only at this node.

pTransId
: Input. XA identifier of the transaction to be heuristically rolled back.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

Usage Notes

Only transactions with a status of *prepared* or *idle* can be rolled back. Once heuristically rolled back, the database manager remembers the state of the transaction until "sqlxhfrg - Forget Transaction Status" on page 449 is issued.

For information about the *SQLXA_XID* structure, see "SQLXA-XID" on page 443.

sqlxphrl - Roll Back an Indoubt Transaction

Appendix C. Precompiler Customization APIs

There is a set of precompiler service APIs which enable the customization of precompilers. Information about what these APIs are, and how to use them, is available electronically as follows:

- CompuServe

  The file is located in the IBM DB2 Family Forum on CompuServe (**GO IBMDB2**). Once in this forum, you can get a file called `PREPAPI.TXT` from Library 1. This file must be downloaded in ASCII format.

- Internet

  An anonymous FTP site containing this information is available. The site is called **ps.boulder.ibm.com**. The file, called `prepapi.txt`, is located in the directory `/ps/products/db2/info`. This file is in ASCII format.

For more generic information about what is available on CompuServe and the Internet, or how to access it, see “Contacting IBM” on page 543.

If you do not have access to either electronic forum and would like to get a copy of this information, you can call IBM Service as described in the *Service Information Flyer*.

Appendix D. Backup and Restore APIs for Vendor Products

DB2 provides interfaces that can be used by third-party media management products to store and retrieve data for backup and restore functions. This functionality is designed to augment the backup and restore data targets of diskette, disk, tape (UNIX based systems only), and ADSM, that are supported as a standard part of DB2.

These third-party media management products will be referred to as vendor products in the remainder of this appendix.

DB2 defines a set of function prototypes that provide a general purpose data interface to backup and restore that can be used by many vendors. These functions are to be provided by the vendor in a shared library on UNIX based systems, or DLL on OS/2 or the Windows operating system. When the functions are invoked by DB2, the shared library or DLL specified by the calling backup or restore routine is loaded and the functions provided by the vendor are called to perform the required tasks.

This appendix is divided into four parts:

- Operational overview of DB2's interaction with vendor products.
- Detailed descriptions of DB2's vendor APIs.
- Information on the data structures used in the API calls.
- Details on invoking backup and restore using vendor products.

Operational Overview

Five functions are defined to interface DB2 and the vendor product:

- sqluvint - Initialize and Link to Device
- sqluvget - Reading Data from Device
- sqluvput - Writing Data to Device
- sqluvend - Unlink the Device
- sqluvdel - Delete Committed Session

DB2 will call these functions, and they should be provided by the vendor product in a shared library on UNIX based systems, or in a DLL on OS/2 or the Windows operating system.

Note: The shared library or DLL code will be run as part of the database engine code. Therefore, it must be reentrant and thoroughly debugged. An errant function may compromise data integrity of the database.

The sequence of functions that DB2 will call in a specific backup or restore session depends on these factors:

- The number of sessions that will be utilized (one or more)?
- Whether it is a backup or a restore.
- The PROMPTING mode that is specified on the backup or restore.
- The characteristics of the device that the data is stored on.
- Any errors encountered during the operation.

Number of Sessions

DB2 supports the backup and restore of database objects using one or more data streams or sessions. A backup or restore using three sessions would require three physical or logical devices to be available. When vendor device support is being used, it is the vendor's functions that are responsible for managing the interface to each physical or logical device. DB2 simply sends or receives data buffers to or from the vendor provided functions.

The number of sessions to be used is specified as a parameter by the application that calls the backup or restore database function. This value is provided in the INIT-INPUT structure used by **sqluvint** (see "sqluvint - Initialize and Link to Device" on page 467).

DB2 will continue to initialize sessions until the specified number is reached, or it receives an SQLUV_MAX_LINK_GRANT warning return code from an **sqluvint** call. In order to warn DB2 that it has reached the maximum number of sessions that it can support, the vendor product will require code to track the number of active sessions. Failure to warn DB2 could lead to a DB2 initialize session request that fails, resulting in a termination of all sessions and the failure of the entire backup or restore operation.

When the operation is backup, DB2 writes a media header record at the beginning of each session. It contains information that DB2 utilizes to identify the session during a restore. DB2 uniquely identifies each session by appending a sequence number to the name of the backup. It starts at 1 (one) for the first session and is incremented by one each time another session is initiated with an **sqluvint** call for a backup or restore operation. For more details, see "INIT-INPUT" on page 483.

When the backup is successfully completed, DB2 writes a media trailer to the last session it closes. This trailer includes information that tells DB2 how many sessions were used to perform the backup. During restore, this information is used to ensure all the sessions, or data streams, have been restored.

Operation with No Errors, Warnings or Prompting

For backup, the following sequence of calls will be issued by DB2 for **each** session.

```
sqluvint, action = SQLUV_WRITE
```

followed by 1 to n

```
sqluvput
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

When DB2 issues an **sqluvend** call (action SQLUV_COMMIT), it expects the vendor product to appropriately save the output data. A return code of SQLUV_OK to DB2 indicates success.

The DB2-INFO structure, used on the **sqluvint** call, contains the information required to identify the backup (see "DB2-INFO" on page 479). A sequence number is supplied.

The vendor product may choose to save this information. DB2 will use it during restore to identify the backup that will be restored.

For restore, the sequence of calls for each session is:

```
sqluvint, action = SQLUV_READ
```

followed by 1 to n

```
sqluvget
```

followed by 1

```
sqluvend, action = SQLUV_COMMIT
```

The information in the DB2-INFO structure used on the **sqluvint** call will contain the information required to identify the backup. Sequence number is not supplied. DB2 expects that all backup objects (session outputs committed during backup) will be returned, and is not sensitive to the order in which they are restored, but does check the media tail to ensure that they have all been processed.

Note: Not all vendor products will keep a record of the names of the backup objects. This is most likely when the backups are being done to tapes, or other media of limited capacity. During the initialization of restore sessions, the identification information can be utilized to stage the necessary backup objects so that they are available when required; this may be most useful when juke boxes or robotic systems are used to store the backups. DB2 will always check the media header (first record in each session's output) to ensure that the correct data is being restored.

PROMPTING Mode

When a backup or restore is initiated, two prompting modes are possible:

- WITHOUT PROMPTING or NOINTERRUPT where there is no opportunity for the vendor product to write messages to the user, or for the user to respond to them.
- PROMPTING or INTERRUPT where the user can receive and respond to messages from the vendor product.

For PROMPTING mode, backup and restore define three possible user responses:

- Continue

  The operation of writing or reading data to the device will resume.

- Device terminate

  The device will receive no additional data and the session is terminated.

- Terminate

  The entire backup or restore operation is terminated.

The use of the PROMPTING and WITHOUT PROMPTING modes is discussed in the sections that follow.

Device Characteristics

For the purposes of the vendor device support APIs, two general types of devices are defined:

- Limited capacity devices requiring user action to change the media, for example, a tape drive, diskette, or CDROM drive.
- Very large capacity devices where normal operations do not require the user be involved with handling media; for example, a juke box, or an intelligent, robotic media handling device.

A limited capacity device may require that the user be prompted to load additional media during the backup or restore operation. Generally DB2 is not sensitive to the order in which the media is loaded for either backup or restore. It also provides facilities to pass vendor media handling messages to the user. This prompting requires that the backup or restore operation be initiated with PROMPTING on. The media handling message text is specified in the description field of the return code structure.

If **PROMPTING** is on and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a **sqluvput** (write) or **sqluvget** (read) call, then DB2 will:

- Mark the last buffer sent to the session to be resent, if the call was **sqluvput**. It will be put to a session later.
- Call the session with **sqluvend** (action = SQLUV_COMMIT). If successful (SQLUV_OK return code), DB2 will:
  - Write a message to the user containing a vendor media handling message from the return code structure that signaled end-of-media.
  - Prompt the user for a continue, device terminate, or terminate response.

Based on the user response, DB2 will:

- If **continue**, DB2 will initialize another session using the **sqluvint** call, and when successful, begin writing data to or reading data from the session. To identify the session uniquely when writing, DB2 increments the sequence number. The sequence number is available in the DB2-INFO structure used with **sqluvint**, and is in the media header record, which is the first data record sent to the session.

  DB2 will not start more sessions than requested when backup or restore is started or indicated by the vendor product with a SQLUV_MAX_LINK_GRANT warning on an **sqluvint**.
- If **device terminate**, DB2 will not attempt to initialize another session, and the number of active session will be reduced by one. DB2 will not allow all sessions to be terminated by device terminate responses; at least one must be kept active until the backup or restore operation completes (for example, all data is processed).
- If **terminate**, DB2 will terminate the backup or restore operation. For more information on exactly what DB2 does to terminate the sessions, see “If Error Conditions Are Returned to DB2” on page 463.

Since the performance of backup or restore is often dependent on the number of devices being used, it is important that parallelism be maintained. For backup, users should be encouraged to respond to the prompting with a continue, unless they know that the remaining active sessions will hold the data that is still to be written out. For restore, users should use the continue response until all media has been processed or is being processed (for example, all the tapes have been read or are being read).

If the backup or restore mode is **WITHOUT PROMPTING** and DB2 receives an SQLUV_ENDOFMEDIA or an SQLUV_ENDOFMEDIA_NO_DATA return code from a session, it will terminate the session and not attempt to open another session. If all sessions return end-of-media to DB2 before the backup or restore is complete, then the backup or restore operation will fail. Because of this, WITHOUT PROMPTING should be used carefully with limited capacity devices. However, it makes sense to operate in this mode with very large capacity devices.

It is possible for the vendor product to hide media mounting and switching actions from DB2, so that the device appears to have infinite capacity. Some very large capacity devices operate in this mode. In these cases, it is critical that all the data that was backed up be returned to DB2 in the same order when a restore operation is in progress. Failure to do so could result in missing data, but DB2 would assume a successful restore operation, since it has no way of detecting the missing data.

DB2 writes data to the vendor product with the assumption that each buffer will be contained on one and only one media (for example, a tape). It is possible for the vendor product to split these buffers across multiple media without DB2's knowledge. In these cases, the order in which the media is processed during a restore is critical, since the vendor product will be responsible for returning reconstructed buffers from the multiple media to DB2. Failure to do so will result in a failure of the restore operation.

If Error Conditions Are Returned to DB2

When performing a backup or restore operation, DB2 expects that all sessions will complete successfully, or the entire backup or restore operation fails. A session signals completed correctly (for example, committed) to DB2 with an SQLUV_OK return code on the call **sqluvend**, action = SQLUV_COMMIT.

If unrecoverable errors are encountered, the session will be terminated by DB2. These can be DB2 errors, or errors returned to DB2 from the vendor product. Since all sessions must commit successfully to have a complete backup or restore, the failure of one will cause DB2 to terminate the other sessions associated with the operation.

If the vendor product decides to respond to a call from DB2 with an unrecoverable return code, the vendor product can optionally provide additional information to the user using message text placed in the description field of the RETURN-CODE structure. This message text will be presented to the user along with the DB2 information, so that corrective action may be taken.

There will be backup scenarios where a session has committed successfully, and another session associated with the backup operation experiences an unrecoverable error. Since all sessions must complete successfully before a backup operation is successful, DB2 must delete the output data in the committed sessions. This is accomplished with the following sequence of calls:

- Using an **sqluvint** call, action = SQLUV_READ, DB2 will initialize a session using the same information originally used to start the committed session.
- The vendor product should verify the existence of the object using the data in the DB2-INFO structure, and return SQLUV_OK if it is found.
- DB2 will then utilize a **sqluvdel** call to request deletion of the object (previously committed output).
- Assuming the **sqluvdel** is successful, an **sqluvend**, action = SQLUV_COMMIT will be issued to terminate the session.

The information in the DB2-INFO structure will contain a valid sequence number during the initialization call to uniquely identify the object (committed session output) to be deleted.

Warning Conditions

It is possible for DB2 to receive warning return codes from the vendor product; for example, under the condition that a device is not ready or some other correctable condition has occurred. This is true for both read and write operations.

On the **sqluvput** and **sqluvget** calls, the vendor can set the return code to SQLUV_WARNING and optionally provide additional information to the user using message text placed in the description field of the return code structure. This message text will be presented to the user, so that corrective action may be taken. Again the user can respond in one of three ways: continue, device terminate, or terminate. The mechanism used to accomplish communication with the user is the same as for end-of-media conditions.

DB2's actions will be:

- For continue, DB2 will attempt to rewrite the buffer using **sqluvput** if the operation is backup. If the operation is restore, DB2 will issue an **sqluvget** call, to read the next buffer.
- For device terminate or terminate, DB2 will terminate the entire backup or restore in the same way that it would for an unrecoverable error (for example, terminate active sessions and delete committed sessions).

Details about possible return codes for each function call and DB2 reactions are specified in the following API sections.

Operational Hints and Tips

This section provides some hints and tips when building vendor products.

Recovery History File

A recovery history file can be used as an aid in database recovery operations. It is associated with each database and is automatically updated with each backup or restore operation. A general overview of the file is provided in the *Administration Guide*. The information in the file can be viewed, updated and pruned through the following facilities:

- Control Center
- Command Line Processor
  - LIST BACKUP/HISTROY
  - PRUNE HISTORY
  - UPDATE RECOVERY HISTORY FILE
- APIs
  - sqluhcls, sqluhgne, slquhops, sqluhprn, and sqluhupd.

For information about the layout of the file, see “SQLUHINFO” on page 426.

When a backup operation completes, a record or records are written to the file. If the output of the backup operation was directed to vendor devices, the DEVICE field in the history record will contain a 0, and the LOCATION field will contain either:

- The vendor file name supplied when the backup was invoked.
- The name of the shared library if there was no vendor file name supplied when the backup was invoked.

See “Invoking Backup/Restore Using Vendor Products” on page 487 for more details about specifying this option. If the vendor file name is not specified, LOCATION will be blank.

The LOCATION field can be updated using any of the above facilities. This capability can be utilized to update the location of the backup information if limited capacity devices (for example, removable media) have been used to hold the backup, and the media is physically moved to a different storage location (for example, off-site). If this is done, then this file can be utilized to assist in locating a backup when a recovery is necessary.

Functions and Data Structures

The following sections describe the generic functions and data structures available for use by the vendor products.

The APIs for vendor products are:

- “sqluvint - Initialize and Link to Device” on page 467
- “sqluvget - Reading Data from Device” on page 471
- “sqluvput - Writing Data to Device” on page 473
- “sqluvend - Unlink the Device and Release its Resources” on page 475
- “sqluvdel - Delete Committed Session” on page 477

The data structures used by the vendor APIs are:

“DB2-INFO” on page 479
: Contains information identifying DB2 to the vendor device.

“VENDOR-INFO” on page 482
: Contains information identifying the vendor and version of the device.

“INIT-INPUT” on page 483
: Sets up a logical link between DB2 and the vendor device.

“INIT-OUTPUT” on page 484
: Contains output from the device.

“DATA” on page 485
: Contains data transferred between DB2 and the vendor device.

“RETURN-CODE” on page 486
: Contains return code and explanation of the error.

sqluvint - Initialize and Link to Device

This function is called to provide information for initialization and establishment of a logical link between DB2 and the vendor device.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

Init_input
: Input. Structure that contains information provided by DB2 to establish a logical link with the vendor device.

Init_output
: Output. Structure that contains the output returned by the vendor device.

Return_code
: Output. Structure that contains the return code to be passed to DB2, and a brief text explanation.

Usage Notes

For each media I/O session, DB2 will call this function to obtain a device handle. If for any reason, the vendor function encounters an error during initialization, it will indicate it via a return code. If the return code indicates an error, DB2 may choose to terminate the operation by calling the **sqluvend** function. Details on possible return codes, and the DB2 reaction to each of these, is contained in the return codes table (see Table 74 on page 468).

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 (see “PROMPTING Mode” on page 461 and “Device Characteristics” on page 462 for more information).

DB2 names the backup being written or the restore to be read via fields in the DB2-INFO structure. In the case of an action = SQLUV_READ, the vendor product must check for the existence of the named object. If it cannot be found, the return code should be set to SQLUV_OBJ_NOT_FOUND so that DB2 will take the appropriate action.

After initialization is completed successfully, DB2 will continue by issuing other data transfer functions, but may terminate the session at any time with an **sqluvend** call.

Return Codes

Table 74 (Page 1 of 3). 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 or sqluvdel (see comments)	If action = SQLUV_WRITE, the next call will be sqluvput (to BACKUP data). If action = SQLUV_READ, verify the existence of the named object prior to returning SQLUV_OK; the next call could be a sqluvget (to RESTORE data) or an sqluvdel (to delete a committed session).
SQLUV_LINK_EXIST	Session activated previously.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

Table 74 (Page 2 of 3). Valid Return Codes for sqluvint and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_COMM_ ERROR	Communication error with device.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_VERSION	The DB2 and vendor products are incompatible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ACTION	Invalid action is requested. This could also be used to indicate that the combination of parameters results in an operation which is not possible.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NO_DEV_ AVAIL	No device is available for use at the moment.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJ_NOT_ FOUND	Object specified cannot be found. This should be used when the action on the sqluvint call is 'R' (read) and the requested object cannot be found based on the criteria specified in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_OBJS_FOUND	More than 1 object matches the specified criteria. This will result when the action on the sqluvint call is 'R' (read) and more than one object matches the criteria in the DB2-INFO structure.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_USERID	Invalid userid specified.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_ PASSWORD	Invalid password provided.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_INV_OPTIONS	Invalid options encountered in the vendor options field.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

Table 74 (Page 3 of 3). Valid Return Codes for sqluvint and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_INIT_FAILED	Initialization failed and the session is to be terminated.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_DEV_ERROR	Device error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_MAX_LINK_ GRANT	Max number of links established.	sqluvput, sqluvget or sqluvdel (see comments)	This is treated as a warning by DB2. The warning tells DB2 not to open additional sessions with the vendor product, because the maximum number of sessions it can support has been reached (note: this could be due to device availability). If action = SQLUV_WRITE (BACKUP), the next call will be sqluvput. If action = SQLUV_READ, you should verify the existence of the named object prior to returning SQLUV_MAX_LINK_GRANT; the next call could be a sqluvget (to RESTORE data) or an sqluvdel (to delete a committed session).
SQLUV_IO_ERROR	I/O error.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.
SQLUV_NOT_ ENOUGH_SPACE	There is not enough space to store the entire backup image; the size estimate is provided as a 64 bit value in bytes.	no further calls	Session initialization fails. Free up memory allocated for this session and terminate. A sqluvend call will not be received, since the session was never established.

sqluvget - Reading Data from Device

After initialization, this function can be called to read data from the device.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data         *,
  struct Return_code  *);
/* ... */
```

API Parameters

pVendorCB
: Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data
: Output. Data buffer filled with data if the function call is successful.

Return_code
: Output. The return code from the API call.

Usage Notes

This is used by the restore function.

Return Codes

Table 75 (Page 1 of 2). 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

Table 75 (Page 2 of 2). Valid Return Codes for sqluvget and Resulting DB2 Action

Literal in Header File	**Description**	**Probable Next Call**	**Other Comments**
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA or SQLUV_ENDOFMEDIA_NO_DATA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvget, or sqluvend, action =SQLU_ABORT	See the explanation of DB2's handling of warnings (“Warning Conditions” on page 464).
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_MORE_DATA	Operation successful; more data available.	sqluvget	
SQLUV_ENDOFMEDIA_NO_DATA	End of media and 0 bytes read (for example, end of tape).	sqluvend	See the explanation of DB2's handling of end-of-media conditions under “PROMPTING Mode” on page 461, and “Device Characteristics” on page 462.
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 as described under “PROMPTING Mode” on page 461, and “Device Characteristics” on page 462.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.

Next call:

- [a] If the next call will be an sqluvend, action = SQLU_ABORT, this session will be terminated. In addition, all other active sessions are terminated with sqluvend, action = SQLU_ABORT.

sqluvput - Writing Data to Device

After initialization, this function can be used to write data to the device.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

pVendorCB
: Input. Pointer to space allocated for the DATA structure (including the data buffer) and Return_code.

Data
: Output. Data buffer filled with data to be written out.

Return_code
: Output. The return code from the API call.

Usage Notes

This is used in the backup function.

Return Codes

Table 76. Valid Return Codes for sqluvput and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvput or sqluvend, if complete (for example, DB2 has no more data)	Inform other processes of successful operation.
SQLUV_COMM_ERROR	Communication error with device.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_ACTION	Invalid action is requested.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_DEV_HANDLE	Invalid device handle.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_INV_BUFF_SIZE	Invalid buffer size specified.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_ENDOFMEDIA	End of media reached, for example, end of tape.	sqluvend	See the explanation of DB2's handling of end-of-media conditions under “PROMPTING Mode” on page 461, and “Device Characteristics” on page 462.
SQLUV_DATA_RESEND	Device requested to have buffer sent again.	sqluvput	DB2 will retransmit the last buffer. This will only be done once.
SQLUV_DEV_ERROR	Device error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_WARNING	Warning. This should not be used to indicate end-of-media to DB2; use SQLUV_ENDOFMEDIA for this purpose. However, device not ready conditions can be indicated using this return code.	sqluvput	See the explanation of DB2's handling of warnings in “Warning Conditions” on page 464.
SQLUV_LINK_NOT_EXIST	No link currently exists.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.
SQLUV_IO_ERROR	I/O error.	sqluvend, action = SQLU_ABORT[a]	The session will be terminated.

Next call:

- [a] If the next call will be an sqluvend, action = SQLU_ABORT, this session will be terminated. In addition, all other active sessions are terminated with sqluvend, action = SQLU_ABORT. Committed sessions are deleted with an sqluvint, sqluvdel, and sqluvend sequence of calls (see “If Error Conditions Are Returned to DB2” on page 463).

sqluvend - Unlink the Device and Release its Resources

Ends or unlinks the device, and frees all its related resources. The vendor has to free or release unused resources before returning to DB2 (for example, allocated space and file handles).

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  long int action,
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

action
: Input. Used to commit or abort the session:
  - SQLU_COMMIT (0 = to commit)
  - SQLU_ABORT (1 = to abort)

pVendorCB
: Input. Pointer to the Init_output structure.

Init_output
: Output. Space for Init_output deallocated. The data has been committed to stable storage for a backup if action is to commit. The data is purged for a backup if the action is to abort.

Return code
: Output. The return code from the API call.

Usage Notes

This function will be called for each session 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 will assume that the output data has been appropriately saved by the vendor's 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 must assume that there are problems with the entire backup or restore. All active sessions will be terminated by **sqluvend** calls with action = SQLUV_ABORT. For a backup operation, committed sessions will receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls (see "If Error Conditions Are Returned to DB2" on page 463).

- Abort

  A problem has been encountered by DB2, and there will be no more reading of data or writing of data to the session.

  For a write (BACKUP) session, the vendor should delete the partial output dataset, and use a SQLUV_OK return code if the partial output is deleted. Also, DB2 assumes that there are problems with the entire backup. All active sessions will be terminated by **sqluvend** calls with action = SQLUV_ABORT, and committed sessions will receive a **sqluvint**, **sqluvdel**, and **sqluvend** sequence of calls (see "If Error Conditions Are Returned to DB2" on page 463).

  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 will terminate all the restore sessions by sqluvend calls with action = SQLUV_ABORT. If the vendor returns SQLUV_ABORT_FAILED to DB2, the caller will not be 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 77. 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.

sqluvdel - Delete Committed Session

Deletes committed sessions.

Authorization

One of the following:

- *sysadm*
- *dbadm*

Required Connection

Database

API Include File

sql.h

C API Syntax

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API Parameters

Init_input
: Input. Space allocated for Init_input and Return_code.

Return_code
: Output. Return code from the API call. The object pointed to by the Init_input structure is deleted.

Usage Notes

If multiple sessions are opened, and some sessions are committed but one of them fails, this function is called to delete each committed session. An **sqluvint** call will precede this call and will be utilized to identify the output data to be deleted. If the return code from this call is a SQLUV_DELETE_FAILED, DB2 will not notify the caller of this error, because DB2 returns the first fatal failure and ignores subsequent failures. In this case, for DB2 to have called **sqluvdel**, an initial fatal error must have occurred.

sqluvdel - Delete Committed Session

Return Codes

Table 78. Valid Return Codes for sqluvdel and Resulting DB2 Action

Literal in Header File	Description	Probable Next Call	Other Comments
SQLUV_OK	Operation successful.	sqluvend	The next call will terminate the session.
SQLUV_DELETE_FAILED	Delete request failed.	sqluvend	The next call will terminate the session.

DB2-INFO

This structure contains information provided by DB2 to identify itself to the vendor device.

Note: All fields are NULL terminated strings.

Table 79 (Page 1 of 2). Fields in the DB2-INFO Structure

Field Name	Data Type	Description
DB2_id	char	An identifier for the DB2 product. Maximum length of string it points to is 8 characters.
version	char	The current version of the DB2 product. Maximum length of 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 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 string it points to is 8 characters.
action	char	Specifies the action to be taken. Maximum length of string it points to is 1 character.
filename	char	The file name used to identify the backup image. If it is NULL, the *server_id, db2instance, dbname*, and *timestamp* will uniquely identify the backup image. Maximum length of string it points to is 255 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of string it points to is 8 characters.
db2instance	char	The db2instance ID. This is the user ID invoking the command. Maximum length of string it points to is 8 characters.
type	char	Specifies the type of backup to be taken. '0' for full database backup and '3' for table space level backup.
dbname	char	The name of the database to be backed up or restored. Maximum length of string it points to is 8 characters.
alias	char	The alias of the database to be backed up or restored. Maximum length of string it points to is 8 characters.
timestamp	char	The time stamp used to identify the backup image. Maximum length of string it points to is 26 characters.
sequence	char	Specifies the file extension for the backup image. It starts at one for the first session and is incremented by one each time another session is initiated with an sqluvint call. Maximum length of string it points to is 3 characters.
obj_list	struct sqlu_gen_list	Lists the objects in the backup image. This is provided to the vendors for their information only.

Table 79 (Page 2 of 2). Fields in the DB2-INFO Structure

Field Name	Data Type	Description
max_bytes_per_txn	long	Specifies to the vendor in bytes, the transfer buffer size specified by the user.
image_filename	char	Reserved for future use.
reserve	void	Reserved for future use.
nodename	char	Name of the node at which the backup was generated.
password	char	Password for the node at which the backup was generated.
owner	char	ID of the backup originator.
mcNameP	char	Management class.
nodeNum	char	Node number.

The *filename,* or *server_id, db2instance, type, dbname* and *timestamp* uniquely identifies the backup image. The sequence number specified by *seq* identifies the file extension. When a backup image is to be restored, the same values must be used to retrieve the backup image. Depending on the vendor product, if *filename* is used, the other parameters may be set to NULL, and vice versa.

Language Syntax

C Structure

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                  *DB2_id;
  char                  *version;
  char                  *release;
  char                  *level;
  char                  *action;
  char                  *filename;
  char                  *server_id;
  char                  *db2instance;
  char                  *type;
  char                  *dbname;
  char                  *alias;
  char                  *timestamp;
  char                  *sequence;
  struct sqlu_gen_list  *obj_list;
  long                  max_bytes_per_txn;
  char                  *image_filename;
  void                  *reserve;
  char                  *nodename;
  char                  *password;
  char                  *owner;
  char                  *mcNameP;
  char                  nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

This structure contains information to identify the vendor and the version of the device being used.

Note: All fields are NULL terminated strings.

Table 80. Fields in the VENDOR-INFO Structure

Field Name	Data Type	Description
vendor_id	char	An identifier for the vendor. Maximum length of string it points to is 64 characters.
version	char	The current version of the vendor product. Maximum length of 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 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 string it points to is 8 characters.
server_id	char	A unique name identifying the server where the database resides. Maximum length of string it points to is 8 characters.
max_bytes_per_txn	long	The maximum supported transfer buffer size. Specified by the vendor in bytes. This is used only if the return code from the vendor initialize function is SQLUV_BUFF_SIZE, indicating an invalid buffer size is specified.
num_objects_in_backup	long	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.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Vendor_info
{
  char      *vendor_id;
  char      *version;
  char      *release;
  char      *level;
  char      *server_id;
  long      max_bytes_per_txn;
  long      num_objects_in_backup;
  void      *reserve;
} Vendor_info;
```

INIT-INPUT

This structure contains information provided by DB2 to set up and to establish a logical link with the vendor device.

Note: All fields are NULL terminated strings.

Table 81. Fields in the INIT-INPUT Structure

Field Name	Data Type	Description
DB2_session	struct DB2_info	A description of the session from the DB2 perspective.
size_options	unsigned short	The length for the options field.
size_HI_order	unsigned long	High order 32 bits of DB size estimate in bytes; total size is 64 bits.
size_LOW_order	unsigned long	Low order 32 bits of DB size estimate in bytes; total size is 64 bits.
num_sessions	unsigned short	Number of sessions requested by the user when backup or restore was invoked.
prompt_lvl	char	Prompting level requested by the user when backup or restore was invoked. Maximum length of string it points to is 1 character.
options	void	This information is passed from the application when the backup or restore function is invoked. This data structure must be flat. In other words, no level of indirection is supported. Note that byte-reversal is not done, and that code page is not checked for this data.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  unsigned long    size_HI_order;
  unsigned long    size_LOW_order;
  unsigned short   num_sessions;
  char             *prompt_lvl;
  void             *options;
  void             *reserve;
} Init_input;
```

INIT-OUTPUT

This structure contains the output returned by the vendor device.

Table 82. Fields in the INIT-OUTPUT Structure

Field Name	Data Type	Description
vendor_session	struct Vendor_info	Contains information to identify the vendor to DB2.
pVendorCB	void	Vendor control block.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

This structure contains data transferred (read and write) between DB2 and the vendor device.

Table 83. Fields in the DATA Structure

Field Name	Data Type	Description
obj_num	long	The sequence number assigned by DB2 during backup.
buff_size	long	The size of the buffer.
actual_buf_size	long	The actual number of bytes sent or received. This must not exceed *buff_size*.
dataptr	void	Pointer to the data buffer. DB2 allocates space for the buffer.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Data
{
  long  obj_num;
  long  buff_size;
  long  actual_buff_size;
  void  *dataptr;
  void  *reserve;
} Data;
```

RETURN-CODE

This structure contains the return code and a short text explanation of the error to be returned to DB2.

Table 84. Fields in the RETURN-CODE Structure

Field Name	Data Type	Description
return_code	long	Return code from the vendor function.
description	char	A short text description of the return code.
reserve	void	Reserved for future use.

Language Syntax

C Structure

```
typedef struct Return_code
{
  long  return_code,
  char  description[60],
  void  *reserve,
} Return_code;
```

The following are the valid return codes accepted from vendor products:

SQLUV_OK	Operation is successful
SQLUV_LINK_EXIST	Session activated previously
SQLUV_COMM_ERROR	Communication error with device
SQLUV_INV_VERSION	The DB2 and vendor products are incompatible
SQLUV_INV_ACTION	Invalid action is requested
SQLUV_NO_DEV_AVAIL	No device is available for use at the moment
SQLUV_OBJ_NOT_FOUND	Object specified cannot be found
SQLUV_OBJS_FOUND	More than 1 object matching specification is found
SQLUV_INV_USERID	Invalid user ID specified
SQLUV_INV_PASSWORD	Invalid password provided
SQLUV_INV_OPTIONS	Invalid options specified
SQLUV_INIT_FAILED	Initialization failed
SQLUV_INV_DEV_HANDLE	Invalid device handle
SQLUV_BUFF_SIZE	Invalid buffer size specified
SQLUV_DATA_RESEND	Device requested that last buffer be sent again

SQLUV_COMMIT_FAILED	Commit request failed
SQLUV_DEV_ERROR	Device error
SQLUV_WARNING	Warning, see return code
SQLUV_LINK_NOT_EXIST	Session not activated previously
SQLUV_MORE_DATA	More data to come
SQLUV_ENDOFMEDIA_NO_DATA	End of media encountered with no data
SQLUV_ENDOFMEDIA	End of media encountered
SQLUV_MAX_LINK_GRANT	Maximum number of links established
SQLUV_IO_ERROR	I/O error encountered
SQLUV_DELETE_FAILED	Delete object fails
SQLUV_INV_BKUP_FNAME	Invalid backup file name provided
SQLUV_NOT_ENOUGH_SPACE	Insufficient space for estimated database size

Invoking Backup/Restore Using Vendor Products

Parameters are available to specify the use of vendor products for backup and restore through these interfaces:

- Database Director backup and restore tools
- Command Line Processor (CLP) BACKUP and RESTORE commands
- Backup and Restore API function calls.

The Database Director

The Database Director is the GUI interface for database administration shipped with DB2. Information on invoking the Database Director is contained in the *Command Reference*.

Its use is documented through help panels provided with the director. These should be reviewed to gain an understanding of the backup and restore tools that are part of the director.

The following parameters are used to specify the use of vendor device support:

To Specify	Database Director Input Variables (for both Backup and Restore)
Use of vendor device and library name	Select *Use Library*, and specify the library name (on UNIX based systems) or the DLL name (on OS/2 or the Windows operating system).
Number of sessions	*Sessions*
Vendor options	not supported
Vendor filename	not supported

To Specify	Database Director Input Variables (for both Backup and Restore)
Transfer buffer size	For backup: *Size of each Buffer* For restore: not applicable.

The Command Line Processor

The command line processor (CLP) is the non-GUI tool shipped with DB2 that can be utilized for database administration and other tasks. The BACKUP DATABASE and RESTORE DATABASE CLP commands are documented in the *Command Reference*.

The specification of vendor device support is handled by the following parameters:

To Specify	Command Line Processor Parameter	
	for Backup	for Restore
Use of vendor device and library name	library-name	shared-library
Number of sessions	num-sessions	num-sessions
Vendor options	not supported	not supported
Vendor file name	not supported	not supported
Transfer buffer size	buffer-size	buffer-size

Backup and Restore API Function Calls

Two API function calls are provided to support backup and restore: **sqlubkup** for backup (see "sqlubkp - Backup Database" on page 230), and **sqlursto** for restore (see "sqlurst - Restore Database" on page 309).

A number of parameters on these API calls support the invocation and passing of data to the vendor device support functions:

To Specify	API Parameter (for both sqlubkup and sqlursto)
Use of vendor device and library name	In structure sqlu_media_list, specify a media-type of SQLU_OTHER_MEDIA, and then in structure sqlu_vendor, specify the shared library or DLL in shr_lib.
Number of sessions	In structure sqlu_media_list, specify sessions.
Vendor options	PVendorOptions
Vendor file name	In structure sqlu_media_list, specify a media-type of SQLU_OTHER_MEDIA, and then in structure sqlu_vendor, specify the file name using filename.
Transfer buffer size	BufferSize

Appendix E. Threaded Applications with Concurrent Access

In the default implementation of threaded applications against a DB2 database, serialization of access to the database is enforced by the database APIs. If one thread performs a database call that is blocked for some reason (that is, the table is already in exclusive use), all other threads will be blocked as well. In addition, all threads within a process share a commit scope. True concurrent access to a database can only be achieved through separate processes, or by using the APIs that are described in this section.

This section describes APIs that can be used to allocate and manipulate separate enviroments (contexts) for the use of database APIs and embedded SQL. Each context is a separate entity, and any connection or attachment using one context is independent of all other contexts (and thus all other connections or attachments within a process). In order for work to be done on a context, it must first be associated with a thread. A thread must always have a context when making database API calls or when using embedded SQL. If these APIs to manipuate contexts are not used, all threads within a process share the same context. If these APIs are used, each thread can have its own context. It will have a separate connection to a database or attachment to an instance, and will have its own commit scope.

Contexts need not be associated with a given thread for the duration of a connection or attachment. One thread can attach to a context, connect to a database, detach from the context, and then a second thread can attach to the context and continue doing work using the already existing database connection. Contexts can be passed around among threads in a process, but not among processes.

If the new APIs are not used, the old behavior is in effect, and existing applications need not change.

Even if the new APIs are used, the following APIs continue to be serialized:

- sqlabndx - Bind
- sqlaprep - Precompile Program
- sqluexpr - Export
- sqluimpr - Import.

The new APIs can be used with embedded SQL and the transaction APIs.

These APIs have no effect (that is, they are no-ops) on platforms that do not support application threading.

Note: CLI automatically uses the new scheme (it creates a new context for each incoming connection), and it is up to the user to disable this explicitly. For more information, see the *CLI Guide and Reference*.

sqleAttachToCtx - Attach to Context

Makes the current thread use a specified context. All subsequent database calls made on this thread will use this context. If more than one thread is attached to a given context, access is serialized for these threads, and they share a commit scope.

Scope

The scope of this API is limited to the immediate process.

Authorization

None

Required Connection

Database

API Include File

sql.h

C API Syntax

```
int sqleAttachToCtx (
void          *pCtx,
void          *reserved,
struct sqlca  *pstSqlca);
```

API Parameters

pCtx

Input. A structure defining the context.

Note: *pCtx* must be a valid context previously allocated by “sqleBeginCtx - Create and Attach to an Application Context” on page 491.

reserved

Reserved for future use.

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqleBeginCtx - Create and Attach to an Application Context

Creates an application context, or creates and then attaches to an application context. More than one application context can be created. Each context has its own commit scope. Different threads can attach to different contexts (see “sqleAttachToCtx - Attach to Context” on page 490). 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

Database

API Include File

sql.h

C API Syntax

```
int sqleBeginCtx (
void          **ppCtx,
long          lOptions,
void          *reserved,
struct sqlca  *pstSqlca);
```

API Parameters

ppCtx
: Output. A structure defining the context. This API allocates a data area out of private memory for the storage of context information.

lOptions
: Input. Valid values are:

  SQL_CTX_CREATE_ONLY
  The context memory will be allocated, but there will be no attachment.

  SQL_CTX_BEGIN_ALL
  The context memory will be allocated, and then a call to “sqleAttachToCtx - Attach to Context” on page 490 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.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqleDetachFromCtx - Detach From Context

Detaches the context being used by the current thread. The context will be detached only if an attach to that context has previously been made.

Scope

The scope of this API is limited to the immediate process.

Authorization

None

Required Connection

Database

API Include File

sql.h

C API Syntax

```
int sqleDetachFromCtx (
void           *pCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API Parameters

pCtx

Input. A structure defining the context.

Note: *pCtx* must be a valid context previously allocated by “sqleBeginCtx - Create and Attach to an Application Context” on page 491.

reserved

Reserved for future use.

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqleEndCtx - Detach and Destroy Application Context

Frees all memory associated with a given context.

Scope

The scope of this API is limited to the immediate process.

Authorization

None

Required Connection

None

API Include File

sql.h

C API Syntax

```
int sqleEndCtx (
void          **pCtx,
long          lOptions,
void          *reserved,
struct sqlca  *pstSqlca);
```

API Parameters

pCtx

Output. A structure defining the context. This API frees a data area in private memory used for the storage of context information.

lOptions

Input. Valid values are:

SQL_CTX_FREE_ONLY

The context memory will be freed only if a prior detach has been done.

Note: *pCtx* must be a valid context previously allocated by "sqleBeginCtx - Create and Attach to an Application Context" on page 491.

SQL_CTX_END_ALL

If necessary, a call to "sqleDetachFromCtx - Detach From Context" on page 493 will be made before the memory is freed.

Note: A detach will be done even if the context is still in use. If this option is used, the *ppCtx* parameter can be NULL, but if passed, it must be a valid context previously allocated by "sqleBeginCtx - Create and Attach to an Application Context" on page 491. A call to "sqleGetCurrentCtx - Get Current Context" on page 496 will be made, and the current context freed from there.

reserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

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, “sqlfxdb - Get Database Configuration” on page 201), it is necessary to detach from the instance using “sqledtin - Detach” on page 115 before calling **sqleEndCtx**.

sqleGetCurrentCtx - Get Current Context

Returns the current context associated with a thread.

Scope

The scope of this API is limited to the immediate process.

Authorization

None

Required Connection

Database

API Include File

sql.h

C API Syntax

```
int sqleGetCurrentCtx (
void           **ppCtx,
void           *reserved,
struct sqlca   *pstSqlca);
```

API Parameters

ppCtx
: Input. A structure defining the context.

reserved
: Reserved for future use.

pSqlca
: Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” on page 355.

sqleInterruptCtx - Interrupt Context

Interrupts the specified context.

Scope

The scope of this API is limited to the immediate process.

Authorization

None

Required Connection

Database

API Include File

sql.h

C API Syntax

```
int sqleInterruptCtx (
void          *pCtx,
void          *reserved,
struct sqlca   *pstSqlca);
```

API Parameters

pCtx

Input. A structure defining the context.

Note: *pCtx* must be a valid context previously allocated by "sqleBeginCtx - Create and Attach to an Application Context" on page 491.

reserved

Reserved for future use.

pSqlca

Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" on page 355.

Usage Notes

During processing, this API:

- Switches to the context that has been passed in
- Sends an interrupt
- Switches to the original context
- Exits.

sqleSetTypeCtx - Set Application Context Type

Sets the application context type. This API should be the first database API called inside an application.

Scope

The scope of this API is limited to the immediate process.

Authorization

None

Required Connection

Database

API Include File

sql.h

C API Syntax

```
int sqleSetTypeCtx (
long   lOptions);
```

API Parameters

lOptions

Input. Valid values are:

SQL_CTX_ORIGINAL
All threads will use the same context, and concurrent access will be blocked. This is the default if none of these APIs is called.

SQL_CTX_MULTI_MANUAL
All threads will use separate contexts, and it is up to the application to manage the contex for each thread. See

- “sqleBeginCtx - Create and Attach to an Application Context” on page 491
- “sqleAttachToCtx - Attach to Context” on page 490
- “sqleDetachFromCtx - Detach From Context” on page 493
- “sqleEndCtx - Detach and Destroy Application Context” on page 494.

The following restrictions/changes apply when this option is used:

- When termination is normal, automatic COMMIT at process termination is disabled. All outstanding transactions are rolled back, and all COMMITs must be done explicitly.

- “sqleintr - Interrupt” on page 140 interrupts all contexts. To interrupt a specific context, use “sqleInterruptCtx - Interrupt Context” on page 497.

Usage Notes

This API must be called *before* any other database call, and only the first call is effective.

Appendix F. DB2 Common Server Log Records

This section describes the structure of the DB2 common server log records returned by "sqlurlog - Asynchronous Read Log" on page 297.

All DB2 common server log records begin with a log manager header. This header includes the total log record size, the log record type, and transaction-specific information. It does not include information about accounting, statistics, traces, or performance evaluation. For more information, see " Log Manager Header" on page 503.

Log records are uniquely identified by a log sequence number (LSN). The LSN represents a relative byte address, within the database log, for the first byte of the log record. It marks the offset of the log record from the beginning of the database log.

The log records written by a single transaction are uniquely identifiable by a field in the log record header. The unique transaction identifier is a six-byte field that increments by one whenever a new transaction is started. All log records written by a single transaction contain the same identifier.

When a transaction performs writable work against a table with DATA CAPTURE CHANGES on, or invokes a log writing utility, the transaction is marked as propagatable. Only propagatable transactions have their transaction manager log records marked as propagatable.

Table 85 (Page 1 of 2). DB2 Common Server Log Records

Data Manager	
"Initialize Table" on page 506	New permanent table creation.
"Import Replace (Truncate)" on page 508	Import replace activity.
"Rollback Insert" on page 508	Rollback row insert.
"Reorg Table" on page 508	REORG committed.
"Create Index, Drop Index" on page 509	Index activity.
"Create Table, Drop Table, Rollback Create Table, Rollback Drop Table" on page 509	Table activity.
"Alter Propagation, Alter Check Pending, Rollback Propagation Change, Rollback Check Pending Change" on page 509	Propagation and pending activity.
"Alter Table Add Columns, Rollback Add Columns" on page 510	Adding columns to existing tables.
"Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record" on page 511	Table record activity.
"Update Record" on page 514	Row updates where storage location not changed.

Table 85 (Page 2 of 2). DB2 Common Server Log Records

Long Field Manager	
"Add Long Field Record" on page 516	Long field record activity.
LOB Manager	
"Insert LOB Data Log Record (AFIM_DATA)" on page 517	Adding LOB data with logging.
"Insert LOB Data Log Record (AFIM_AMOUNT)" on page 517	Adding LOB data without logging.
Transaction Manager	
"Normal Commit" on page 518	Transaction commits.
"Heuristic Commit" on page 518	Indoubt transaction commits.
"MPP Coordinator Commit" on page 519	Transaction commits. This is written on a coordinator node for an application that performs updates on at least one subordinator node.
"MPP Subordinator Commit" on page 519	Transaction commits. This is written on a subordinator node.
"Normal Abort" on page 519	Transaction aborts.
"Heuristic Abort" on page 520	Indoubt transaction aborts.
"Local Pending List" on page 520	Transaction commits with a pending list existing.
"Global Pending List" on page 520	Transaction commits (two-phase) with a pending list existing.
"XA Prepare" on page 521	XA transaction preparation in two-phase commit environments.
"MPP Subordinator Prepare" on page 521	MPP transaction preparation in two-phase commit environments. This log record only exists on subordinator nodes.
Utility Manager	
"Migration Begin" on page 522	Catalog migration starts.
"Migration End" on page 523	Catalog migration completes.
"Load Start" on page 523	Table load starts.
"Load Pending List" on page 523	Table load completes.
"Backup End" on page 524	Backup activity completes.
"Tablespace Rolled Forward" on page 524	Table space rollforward completes.
"Tablespace Roll Forward to PIT Begins" on page 524	Marks the beginning of a table space rollforward to a point in time.
"Tablespace Roll Forward to PIT Ends" on page 524	Marks the end of a table space rollforward to a point in time.

Log Manager Header

All DB2 common server 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 86 (Page 1 of 2). Log Manager Log Record Header (LogManagerLogRecordHeader)

Description	Type	Offset (Bytes)
Length of the entire log record	int	0(4)
Type of log record[a]	short	4(2)
Log record general flag[b]	short	6(2)
Log Sequence Number of the previous log record written by this transaction. It is used to chain log records by transaction. If the value is 0000 0000 0000, this is the first log record written by the transaction.	SQLU_LSN[c]	8(6)
Unique transaction identifier	SQLU_TID[d]	14(6)
Log Sequence Number of the log record for this transaction prior to the log record being compensated. (Note: For compensation log records only.)	SQLU_LSN	20(6)
Log Sequence Number of the log record for this transaction being compensated. (Note: For propagatable compensation log records only.)	SQLU_LSN	26(6)
Total Length for Log Manager Log Record Header: • *Non Compensation: 20 bytes* • *Compensation: 26 bytes* • *Propagatable Compensation: 32 bytes*		

Table 86 (Page 2 of 2). Log Manager Log Record Header (LogManagerLogRecordHeader)

Description	Type	Offset (Bytes)

Definitions and Values:

[a] Valid log record types

```
A  Normal Abort                p  Tablespace roll forward to PIT starts
c  MPP coordinator commit      q  Tablespace roll forward to PIT ends
C  Compensation                Q  Global Pending List
D  Tablespace Rolled Forward   R  Redo
E  Local Pending List          s  MPP subordinate commit
G  Load Pending List           U  Undo
I  Heuristic Abort             V  Migration Begin
J  Load Start                  W  Migration End
M  Normal Commit               X  Prepare
N  Normal                      Y  Heuristic Commit
o  Backup Start                z  MPP prepare
O  Backup End                  Z  XA prepare
```

[b] Log record general flag constants

```
Redo Always                0x0001
Propagatable               0x0002
Conditionally Recoverable  0x0080
```

[c] Log Sequence Number (LSN)

```
A unique log record identifier representing the relative byte address
of the log record within the database log.

SQLU_LSN: union {  char  [6] ;
                   short [3] ;
                  }
```

[d] Transaction Identifier (TID)

```
A unique log record identifier representing the transaction.

SQLU_TID: union {  char  [6] ;
                   short [3] ;
                  }
```

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 87 (Page 1 of 2). DMS Log Record Header Structure (DMSLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=1)	unsigned char	0(1)

Table 87 (Page 2 of 2). DMS Log Record Header Structure (DMSLogRecordHeader)

Description	Type	Offset (Bytes)
Function identifier[a]	unsigned char	1(1)
Table identifiers Tablespace identifier Table identifier	unsigned short	2(2)
Total Length: 6 bytes		

Definitions and Values:

[a] Valid function identifier values

SQLD_MIN_DP	100	MIN DBMS LOG FUNCTION ID
SQLD_MAX_DP	149	MAX DBMS LOG FUNCTION ID
ADDCOLUMNS_DP	102	Add columns via alter tbl
CRNEWPG_DP	103	Create new page
UNDOADDCOLUMNS_DP	104	Undo add columns
ALTERPROP_DP	105	Alter prop flag
DELREC_DP	106	Delete record on page
UNDOALTERPROP_DP	107	Undo alter prop flag
ALTERPENDING_DP	108	Alter check pending flag
ALTERDEFAULTS_DP	109	Alter user defaults add flag
UNDOADD_DP	110	Undo add a record
UNDODEL_DP	111	Undo delete a record
UNDOUPDT_DP	112	Undo update a record
CRSYSPGR_DP	114	Initialize sys page DTR
REORGPAGE_DP	117	Reorg page
INSREC_DP	118	Insert record on page
UPDREC_DP	120	Update record on
UPDCHGONLY_DP	121	Log only updated
CREATEPERM_DP	128	Initialize a DAT object
UNDOALTERDEFAULTS_DP	131	Undo alter user default flag
UNDOALTERPENDING_DP	132	Undo alter pending flag

Table 88. DOM Log Record Header Structure (DOMLogRecordHeader)

Description	Type	Offset (Bytes)
Component identifier (=4)	unsigned char	0(1)
Function identifier[a]	unsigned char	1(1)
Object identifiers Tablespace identifier Object identifier	unsigned short	2(2)
Table identifiers Tablespace identifier Table identifier	unsigned short	6(2)
Object type	unsigned char	10(1)
Flags	unsigned char	11(1)
Total Length: 12 bytes		

[a] For a list of valid function identifier values, see Table 87 on page 504.

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

The initialize table log record is written when a new permanent table is being created; it signifies table initialization. This record appears after any log records that create the DATA storage object, and before any log records that create the LF and LOB storage objects. This is a Redo log record.

Table 89 (Page 1 of 2). Initialize Table Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
File create LSN	SQLU_LSN	6(6)
Table directory record	variable	12(72)
record type	unsigned char	12(1)
reserved	char	13(1)
index flag	unsigned short	14(2)
index root page	unsigned long	16(4)
TDESC recid	long	20(4)
reserved	char	24(56)
flags[a]	unsigned long	80(4)
Table description length		84(4)
Table description record	variable	88(variable)
record type	unsigned char	88(1)
reserved	char	89(1)
number of columns	unsigned short	90(2)
array	variable long	92(variable)
Total Length: 88 bytes plus table description record length		
Note: [a] Bit 0x00000020 indicates that the table was created with the NOT LOGGED INITIALLY option, and that no DML activity on this table is logged until the transaction that created the table has been committed.		

Table 89 (Page 2 of 2). Initialize Table Log Record Structure

Description	Type	Offset (Bytes)

Table Description Record Details:

column descriptor array

(number of columns) * 8, where each element of the array contains:

- field type[b] (unsigned short, 2 bytes)
- length (2 bytes)
  - If BLOB, CLOB, or DBCLOB, this field is not used. For the maximum length of this field, see the array that follows the column descriptor array.
  - If not DECIMAL, length is the maximum length of the field (short).
  - If PACKED DECIMAL: Byte 1, unsigned char, precision (total length) Byte 2, unsigned char, scale (fraction digits).
- null flag[c] (unsigned short, 2 bytes)
- field offset (unsigned short, 2 bytes) This is the offset from the start of the formatted record to where the field's fixed value can be found.

LOB descriptor array

(number of LOB, CLOB, and DBCLOB fields) * 12, where each element of the array contains:

- length (MAX LENGTH OF FIELD, unsigned long, 4 bytes)
- reserved (internal, unsigned long, 4 bytes)
- log flag (IS COLUMN LOGGED, unsigned long. 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.

[b] field type

SMALLINT	0x0000
INTEGER	0x0001
DECIMAL	0x0002
DOUBLE	0x0003
REAL	0x0004
CHAR	0x0100
VARCHAR	0x0101
LONG VARCHAR	0x0104
DATE	0x0105
TIME	0x0106
TIMESTAMP	0x0107
BLOB	0x0108
CLOB	0x0109
GRAPHIC	0x0200
VARGRAPH	0x0201
LONG VARG	0x0202
DBCLOB	0x0203

[c] null flag

- mutually exclusive: allows nulls, or does not allow nulls
- valid options: no default, type default, or user default

ISNULL	0x01
NONULLS	0x02
TYPE_DEFAULT	0x04
USER_DEFAULT	0x08

Data Manager Log Records

Import Replace (Truncate)

The import replace (truncate) log record is written when an IMPORT REPLACE action is being executed. This record indicates the reinitialization of the table (no user records, new life LSN). The second set of pool and object IDs in the log header identify the table being truncated (IMPORT REPLACE). This is a Redo log record.

Table 90. Import Replace (Truncate) Log Record Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
internal	variable	12(variable)
Total Length: 12 bytes plus variable length		

Rollback Insert

The rollback insert log record is written when an insert row action (INSERT RECORD) is rolled back. This is a Compensation log record.

Table 91. Rollback Insert Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
RID	long	8(4)
Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
Total Length: 16 bytes		

Reorg Table

The reorg table log record is written when the REORG utility has committed to completing the reorganization of a table. This is a Normal log record.

Table 92. Reorg Table Log Record Structure

Description	Type	Offset (Bytes)
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(252)
Index token[a]	unsigned short	2(264)
Temporary tablespace ID[b]	unsigned short	2(266)
Total Length: 268 bytes		
Note: [a] If not 0, it is the index by which the reorg is clustered (clustering index). [b] If not 0, it is the temporary table space that was used to build the reorg.		

Create Index, Drop Index

These log records are written when indexes are created or dropped. The two elements of the log record are:

- The index root page, which is an internal identifier
- The index token, which is equivalent to the `IID` column in SYSIBM.SYSINDEXES. If the value for this element is 0, the log record represents an action on an internal index, and is not related to any user index.

This is a Undo log record.

Table 93. Create Index, Drop Index Log Records Structure

Description	**Type**	**Offset (Bytes)**
Log header	DOMLogRecordHeader	0(12)
Padding	char[ ]	12(2)
Index token	unsigned short	14(2)
Index root page	unsigned long	16(4)

Total Length: 20 bytes

Create Table, Drop Table, Rollback Create Table, Rollback Drop Table

These log records are written when the DATA object for a permanent table is created or dropped. The DATA object is created during a CREATE TABLE, and prior to table initialization (Initialize Table). Create table and drop table are Normal log records. Rollback create table and rollback drop table are Compensation log records.

Table 94. Create Table, Drop Table, Rollback Create Table, Rollback Drop Table Log Records Structure

Description	**Type**	**Offset (Bytes)**
Log header	DOMLogRecordHeader	0(12)
Internal	variable	12(56)

Total Length: 68 bytes

Alter Propagation, Alter Check Pending, Rollback Propagation Change, Rollback Check Pending Change

The alter check pending log record is written when the state of a table is changed as a result of adding or validating constraints. A table is in CHECK PENDING STATE when the flag value for this PENDING state is 1 (TRUE = CHECK PENDING). Access to a table is restricted when it is in the CHECK PENDING state.

The alter propagation log record is written when the user changes the propagation state of a table with an ALTER TABLE statement. Valid flag values are:

- 0 (false = propagation off)
- 1 (true = propagation on).

Alter propagation and alter check pending are Normal log records. Rollback propagation change and rollback check pending change are Compensation log records.

Table 95. Alter Propagation, Alter Check Pending, Rollback Propagation Change, Rollback Check Pending Change Log Records Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
Old flag value	int	8(4)
New flag value	int	12(4)
Total Length: 16 bytes		

Alter Table Add Columns, Rollback Add Columns

The alter table add columns log record is written when the user is adding columns to an existing table using an ALTER TABLE statement. Complete information on the old columns and the resulting columns (new columns equals resulting columns minus old columns) is logged.

- Column count elements represent the old number of columns and the new total number of columns (new or added columns equals new columns minus old columns).
- LOB count elements are used internally. They represent the number of BLOB, CLOB, and DBCLOB fields.
- VAR flag elements are used internally. They indicate whether any fields are of variable length.
- The parallel arrays contain information about the columns defined in the table. The old parallel array defines the table prior to the ALTER TABLE statement, while the new parallel array defines the table resulting from ALTER TABLE statement.
- Each parallel array consists of:
  - An array equivalent to the column descriptor array in the table description record (see “Initialize Table” on page 506).
  - A second array equivalent to the LOB descriptor array in the table description record. However, since this array is parallel to the first, the only elements used are those whose corresponding element in the first array are of type BLOB, CLOB, or DBCLOB.

Alter table add columns is a Normal log record. Rollback add columns is a Compensation log record.

Table 96 (Page 1 of 2). Alter Table Add Columns, Rollback Add Columns Log Records Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordheader	0(6)

Table 96 (Page 2 of 2). Alter Table Add Columns, Rollback Add Columns Log Records Structure

Description	Type	Offset (Bytes)
Padding	char[ ]	6(2)
Old column count	int	8(4)
New column count	int	12(4)
Old LOB count	int	16(4)
New LOB count	int	20(4)
Old LF count	int	24(4)
New LF count	int	28(4)
Old VAR flag value	int	32(4)
New VAR flag value	int	36(4)
Old parallel arrays[a]	variable	40(variable)
New parallel arrays[b]	variable	variable

*Total Length: 40 bytes plus 2 sets of parallel arrays; array size is (old/new column count) * 20.*

Array Elements:

[a] Each element in this array is 8 bytes long.
[b] Each element in this array is 12 bytes long.

For information about the column descriptor array or the LOB descriptor array, see Table 89 on page 506).

Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record

These log records are written when rows are inserted into or deleted from a table. Insert record and delete record log records are generated during an update if the location of the record being updated must be changed to accommodate the modified record data. Insert record and delete record are Normal log records. Rollback delete record and rollback update record are Compensation log records.

Table 97 (Page 1 of 2). Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record Log Records Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
RID	long	8(4)
Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
Record offset	unsigned short	16(2)
Record header and data	variable	18(variable)

Table 97 (Page 2 of 2). Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record Log Records Structure

Description	Type	Offset (Bytes)
Total Length: 18 bytes plus Record length		

Record Header and Data Details:

Record header
4 bytes

- Record type[a] (unsigned char, 1 byte). Records are one of two classes:
  - Updatable
  - Special control

  A value of 0 or 4 indicates that the record can be viewed.

  Each class has three types:
  - Normal
  - Pointer
  - Overflow
- Reserved (char, 1 byte)
- Record length (unsigned short, 2 bytes)

Record
variable
- Record type (unsigned char, 1 byte). Updatable records are one of two types:
  - Internal control
  - Formatted user data

  A value of 1 signifies a formatted user data record.
- Reserved (char, 1 byte)
- The rest of the record is dependent upon the record type and the table descriptor record defined for the table. If the record type is internal control, the data cannot be viewed. The following fields apply to user data records:
  - Fixed length (unsigned short, 2 bytes). This is the length of all fixed portions of the data row.
  - Formatted record (fixed and variable length). For more information about formatted records, see "Formatted User Data Record".

[a] Record data can only be viewed if the record type (specified in the record header) is updatable (that is, *not* special control).

Formatted User Data Record

The formatted record can be a combination of fixed and variable length data. All fields contain a fixed length portion. In addition, there are seven field types that have variable length parts:

- VARCHAR
- LONG VARCHAR
- BLOB

- CLOB
- VARGRAPHIC
- LONG VARG
- DBCLOB

Field Lengths

The length of the fixed portion of the different field types can be determined as follows:

- DECIMAL

  This field is a standard packed decimal in the form: *nnnnnn...s*. The length of the field is: (precision + 2)/2. The sign nibble (s) is xC for positive (+), and xD or xB for negative (−).

- SMALLINT INTEGER 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.

Note: For element addresses, see Table 89 on page 506.

For more detailed information about field types, see the *SQL Reference*.

The following sections describe the location of the fixed portion of each field within the formatted record.

Table Descriptor Record

The table descriptor record describes the column format of the table. It contains an array of column structures, whose elements represent field type, field length, null flag, and field offset. The latter is the offset from the beginning of the formatted record, where the fixed length portion of the field is located.

Table 98. Table Descriptor Record Structure

Table Descriptor Record			
record type	number of columns	column structure • field type • length • null flag • field offset	LOB information
Note: For more information, see Table 89 on page 506.			

For columns that are nullable (as specified by the null flag), there is an additional byte following the fixed length portion of the field. This byte contains one of two values:

- NOT NULL (0x00)
- NULL (0x01)

If the null flag within the formatted record for a column that is nullable is set to 0x00, there is a valid value in the fixed length data portion of the record. If the null flag value is 0x01, the data field value is NULL.

The formatted user data record contains the table data that is visible to the user. It is formatted as a fixed length record, followed by a variable length section.

Table 99. Formatted User Data Record Structure

Formatted User Data Record			
record type	length of fixed section	fixed length section	variable data section
Note: For more information, see Table 97 on page 511.			

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.

Update Record

The update record log record is written when a row is updated, and if its storage location does not change. There are two available log record formats; they are identical to the insert record and the delete record log records (see “Insert Record, Delete Record, Rollback Delete Record, Rollback Update Record” on page 511). One contains the *pre*-update image of the row being updated; the other contains the *post*-update image of the row being updated. This is a Normal log record.

Table 100. Update Record Log Record Structure

Description	Type	Offset (Bytes)
Log header	DMSLogRecordHeader	0(6)
Padding	char[ ]	6(2)
RID	long	8(4)
New Record length	unsigned short	12(2)
Free space	unsigned short	14(2)
Record offset	unsigned short	16(2)
Old record header and data	variable	18(variable)
Log header	DMSLogRecordHeader	variable(6)
Padding	char[ ]	variable(2)
RID	long	variable(4)
Old record length	unsigned short	variable(2)
Free space	unsigned short	variable(2)
Record offset	unsigned short	variable(2)
New record header and data	variable	variable(variable)

Total Length: 36 bytes plus 2 Record lengths

Long Field Manager Log Records

Long field manager log records are written only if a database is configured with LOG RETAIN on or USEREXITS enabled. They are written whenever long field data is inserted into a table. 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 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.

Table 101 (Page 1 of 2). Long Field Manager Log Record Header (LongFieldLogRecordHeader)

Description	Type	Offset (Bytes)
Originator code (component identifier = 3)	unsigned char	0(1)
Operation type	unsigned char	1(1)

Table 101 (Page 2 of 2). Long Field Manager Log Record Header (LongFieldLogRecordHeader)

Description	Type	Offset (Bytes)
Pool identifier	unsigned short	2(2)
Object identifier	unsigned short	4(2)
Parent pool identifier[a]	unsigned short	6(2)
Parent object identifier[b]	unsigned short	8(2)
Total Length: 10 bytes		

Note:

[a] Pool ID of the data object
[b] Object ID of the data object

Add Long Field Record

This log record is written whenever long field data is inserted. The length of the data is rounded up to the next 512-byte boundary.

Table 102. Add Long Field Record Log Record Structure

Description	Type	Offset (Bytes)
Log header	LongFieldLogRecordHeader	0(10)
Long field length[a]	unsigned short	10(2)
File offset[b]	unsigned long	12(4)
Long field data	char[ ]	16(variable)

Note:

[a] Long field data length in 512-byte sectors (actual data length is not logged)
[b] 512-byte sector offset into long field object where data is to be inserted

LOB Manager Log Records

LOB manager log records are written only if a database is configured with LOG RETAIN on or USEREXITS enabled. The log records are written whenever LOB data is inserted into a table. When LOB data is updated, the update is treated as a delete of the old LOB value, followed by an insert of the new value. If the LOB manager is able to determine that the new value is simply the old value with new data appended to it, the new data is appended to the old data. In this case, only the new data is logged.

For LOB columns that were created with the NOT LOGGED option, a log record is still written if the database is forward recoverable. However, instead of logging the actual data, only the quantity of data and its position within the LOB object are logged. During forward recovery, zeros (not user data) are written to the LOB object.

For any LOB value inserted, multiple LOB records may be written. A single LOB record will not contain more than 32 768 bytes of data.

In order to conserve log space, LOB data inserted into tables is not logged if the database is configured for circular logging. In addition, when a LOB value is updated, the before image is shadowed and not logged.

All LOB manager log records begin with a log record header.

All LOB manager log record offsets are from the end of the log manager log record header.

Table 103. LOB Manager Log Record Header Structure

Description	Type	Offset (Bytes)
Originator code (component identifier = 5)	unsigned char	0(1)
Operation identifier	unsigned char	1(1)
Pool identifier	unsigned short	2(2)
Object identifier	unsigned short	4(2)
Parent pool identifier	unsigned short	6(2)
Parent object identifier	unsigned short	8(2)
Object type	unsigned char	10(1)
Total Length: 11 bytes		

Insert LOB Data Log Record (AFIM_DATA)

This log record is written when LOB data is inserted into a LOB column, or appended to an existing LOB value, and logging of the data has been specified.

Table 104. Insert LOB Data Log Record (AFIM_DATA)

Description	Type	Offset (Bytes)
Log header	LOBLogRecordHeader	0(11)
Padding	char	11(1)
Data length	unsigned long	12(4)
Byte address in object	double	16(8)
LOB data	variable	24(variable)
Total Length: 24 bytes plus LOB data		

Insert LOB Data Log Record (AFIM_AMOUNT)

This log record is written instead of the AFIM_DATA log record if logging for the LOB column has been turned off.

Table 105 (Page 1 of 2). Insert LOB Data Log Record (AFIM_AMOUNT)

Description	Type	Offset (Bytes)
Log header	LOBLogRecordHeader	0(11)

Table 105 (Page 2 of 2). Insert LOB Data Log Record (AFIM_AMOUNT)		
Description	**Type**	**Offset (Bytes)**
Padding	char	11(1)
Data length	unsigned long	12(4)
Byte address in object	double	16(8)
Total Length: 24 bytes		

Transaction Manager Log Records

The transaction manager produces log records signifying the completion of transaction events (for example, commit or rollback). The time stamps in the log records are in Coordinated Universal Time (CUT), and mark the time (in seconds) since January 01, 1970.

Normal Commit

This log record is written for XA transactions in a single-node environment, or on the coordinator node in MPP. It is only used for XA applications. The log record is written when a transaction commits after one of the following events:

- A user has issued a COMMIT
- An implicit commit occurs during a CONNECT RESET.

Table 106. Normal Commit Log Record Structure		
Description	**Type**	**Offset (Bytes)**
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	unsigned long	20(4)
Authorization identifier of the application[a]	char [ ]	24(9)
Total Length: 33 bytes propagatable (24 bytes non-propagatable)		
Note: [a] If the log record is marked as propagatable		

Heuristic Commit

This log record is written when an indoubt transaction is committed.

Table 107 (Page 1 of 2). Heuristic Commit Log Record Structure		
Description	**Type**	**Offset (Bytes)**
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	unsigned long	20(4)
Authorization identifier of the application[a]	char [ ]	24(9)
Total Length: 33 bytes propagatable (24 bytes non-propagatable)		

Table 107 (Page 2 of 2). Heuristic Commit Log Record Structure

Description	Type	Offset (Bytes)
Note: [a] If the log record is marked as propagatable		

MPP Coordinator Commit

This log record is written on a coordinator node for an application that performs updates on at least one subordinator node.

Table 108. MPP Coordinator Commit Log Record Structure

Description	Type
Log header	LogManagerLogRecordHeader
MPP identifier of the transaction	SQLP_GXID
Maximum node number	2 bytes
TNL	variable, (max node no / 8) + 1
Authorization identifier	char [ ]

MPP Subordinator Commit

This log record is written on a subordinator node in MPP.

Table 109. MPP Subordinator Commit Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
MPP identifier of the transaction	SQLP_GXID	20(20)
Authorization identifier	char [ ]	40(9)
Total Length: 49 bytes		

Normal Abort

This log record is written when a transaction aborts after one of the following events:

- A user has issued a ROLLBACK
- A deadlock occurs
- An implicit rollback occurs during crash recovery
- An implicit rollback occurs during ROLLFORWARD recovery.

Table 110 (Page 1 of 2). Normal Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier of the application[a]	char [ ]	20(9)

Table 110 (Page 2 of 2). Normal Abort Log Record Structure

Description	Type	Offset (Bytes)
Total Length: 29 bytes propagatable (20 bytes non-propagatable)		
Note: [a] If the log record is marked as propagatable		

Heuristic Abort

This log record is written when an indoubt transaction is aborted.

Table 111. Heuristic Abort Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Authorization identifier of the application[a]	char [ ]	20(9)
Total Length: 29 bytes propagatable (20 bytes non-propagatable)		
Note: [a] If the log record is marked as propagatable		

Local Pending List

This log record is written if a transaction commits and a pending list exists. The pending list is a linked list of nonrecoverable operations (such as deletion of a file) that can only be performed when the user/application issues a COMMIT. The variable length structure contains the pending list entries.

Table 112. Local Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	unsigned long	20(4)
Authorization identifier of the application[a]	char [ ]	24(9)
Pending list entries	variable	33(variable)
Total Length: 33 bytes plus pending list entries propagatable (24 bytes plus pending list entries non-propagatable)		
Note: [a] If the log record is marked as propagatable		

Global Pending List

This log record is written if a transaction involved in a two-phase commit commits, and a pending list exists. The pending list contains nonrecoverable operations (such as deletion of a file) that can only be performed when the user/application issues a COMMIT. The variable length structure contains the pending list entries.

Table 113. Global Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	unsigned long	20(4)
Authorization identifier of the application[a]	char [ ]	24(9)
Global pending list entries	variable	32(variable)
Total Length: 33 bytes plus pending list entries propagatable (24 bytes plus pending list entries non-propagatable)		
Note: [a] If the log record is marked as propagatable		

XA Prepare

This log record is written for XA transactions in a single-node environment, or on the coordinator node in MPP. It is only used for XA applications. The log record is written to mark the preparation of the transaction as part of a two-phase commit. The XA prepare log record describes the application that started the transaction, and is used to recreate an indoubt transaction.

Table 114. XA Prepare Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Log space used by transaction	unsigned long	20(4)
XA identifier of the transaction	variable	24(140)
Application name	char [ ]	164(20)
Application identifier	char [ ]	184(32)
Sequence number	char [ ]	216(4)
Authorization identifier	char [ ]	220(8)
Database alias used by client	char [ ]	228(20)
Code page identifier	unsigned long	248(4)
Time transaction prepared	unsigned long	252(4)
Synclog information	variable	256(variable)
Total Length: 256 bytes plus variable		

MPP Subordinator Prepare

This log record is written for MPP transactions on subordinator nodes. The log record is written to mark the preparation of the transaction as part of a two-phase commit. The MPP subordinator prepare log record describes the application that started the transaction, and is used to recreate an indoubt transaction.

Table 115. MPP Subordinator Prepare Log Record Structure		
Description	**Type**	**Offset (Bytes)**
Log header	LogManagerLogRecordHeader	0(20)
Log space used by transaction	unsigned long	20(4)
Coordinator LSN	unsigned char	24(6)
MPP identifier of the transaction	SQLP_GXID	30(20)
Application name	char [ ]	50(20)
Application identifier	char [ ]	70(32)
Sequence number	char [ ]	102(4)
Authorization identifier	char [ ]	106(8)
Database alias used by client	char [ ]	114(20)
Code page identifier	unsigned long	134(4)
Time transaction prepared	unsigned long	138(4)
Total Length: 142 bytes		

Utility Manager Log Records

The utility manager produces log records associated with the following DB2 common server utilities:

- Migration
- Load
- Backup
- Table space rollforward.

The log records signify the beginning or the end of the requested activity. All utility manager log records are marked as propagatable regardless of the tables that they affect.

Migration Begin

This log record is associated with the beginning of catalog migration.

Table 116. Migration Begin Log Record Structure		
Description	**Type**	**Offset (Bytes)**
Log header	LogManagerLogRecordHeader	0(20)
Migration start time	char[ ]	20(10)
Migrate from release	unsigned short	30(2)
Migrate to release	unsigned short	32(2)
Total Length: 34 bytes		

Migration End

This log record is associated with the successful completion of catalog migration.

Table 117. Migration End Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Migration end time	char[ ]	20(10)
Migrate to release	unsigned short	30(2)
Total Length: 32 bytes		

Load Start

This log record is associated with the beginning of a load.

Table 118. Load Start Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Log record identifier	unsigned long	20(4)
Pool identifier	unsigned short	24(2)
Object identifier	unsigned short	26(2)
Flag	unsigned char	28(1)
Object pool list	variable	29(variable)
Total Length: 29 bytes plus variable		

Load Pending List

This log record is written when a load transaction commits. The pending list is a linked list of nonrecoverable operations which are deferred until the transaction commits. No commit log record follows this transaction.

Table 119. Load Pending List Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Time transaction committed	unsigned long	20(4)
Authorization identifier of the application[a]	char[ ]	24(9)
Pending list entries	variable	33(variable)
Total Length: 33 bytes plus pending list entries propagatable (24 bytes plus pending list entries non-propagatable)		
Note: [a] If the log record is marked as propagatable		

Utility Manager Log Records

Backup End

This log record is associated with the end of a successful backup.

Table 120. Backup End Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Backup end time	unsigned long	20(4)
Total Length: 24 bytes		

Tablespace Rolled Forward

This log record is associated with table space ROLLFORWARD recovery. It is written for each table space that is successfully rolled forward.

Table 121. Table Space Rolled Forward Log Record Structure

Description	Type	Offset (Bytes)
Log header	LogManagerLogRecordHeader	0(20)
Table space identifier	unsigned short	20(2)
Total Length: 22 bytes		

Tablespace Roll Forward to PIT Begins

This log record is associated with table space ROLLFORWARD recovery. It marks the beginning of a table space rollforward to a point in time.

Table 122. Table Space Roll Forward to PIT Begins Log Record Structure

Description	Type	Offset (Bytes)
Time stamp to which table spaces are being rolled forward.	unsigned long	0(4)
Time stamp for this log record.	unsigned long	4(4)
Number of pools being rolled forward.	unsigned short	8(2)
Integer list of pool IDs that are being rolled forward.	int*numpools	10(variable)
Total Length: 10 bytes plus variable		

Tablespace Roll Forward to PIT Ends

This log record is associated with table space ROLLFORWARD recovery. It marks the end of a table space rollforward to a point in time.

Table 123. Table Space Roll Forward to PIT Ends Log Record Structure

Description	**Type**	**Offset (Bytes)**
Time stamp for this log record.	unsigned long	0(4)
Time stamp to which table spaces were rolled forward.	unsigned long	4(4)
A flag whose value is TRUE if the roll forward was successful, or FALSE if the roll forward was canceled.	int	8(4)
Total Length: 12 bytes		

Utility Manager Log Records

Appendix G. Application Migration Considerations

This section describes issues that should be considered before migrating an application to Version 5.

There are four possible operating scenarios:

1. Running pre-Version 5 applications against databases that have not been migrated
2. Running pre-Version 5 applications against migrated databases
3. Updating applications with Version 5 APIs
4. Running Version 5 applications against migrated databases.

The first and the fourth are consistent operating environments that do not require qualification.

The second, in which only the databases have been migrated, should work without changes to any application, because back-level applications are supported. However, as with any new version, a small number of incompatibilities can occur, and these are described in the *Administration Guide*.

For the third scenario, in which applications are to be updated with Version 5 APIs, the following points should be considered:

- All pre-Version 5 APIs that have been discontinued in Version 5 are still defined in the Version 5 header files, so that older applications will compile and link with Version 5 headers.
- Discontinued APIs should be removed from applications as soon as possible to enable these applications to take full advantage of the new functions available in Version 5, and to position the applications for future enhancements.
- The names of the APIs listed below have changed because of new functionality in Version 5. Users should scan for these names in their application source code to identify the changes required following Version 5 migration of the application.

  APIs that are not listed do not require changes following migration of an application.

  Note that an application may contain the generic version of an API call, depending on the application programming language being used. In all cases, the generic version of the API name is identical to the C version of the name, with the exception that the fourth character is always **g**.

Changed APIs and Data Structures

Table 124. Discontinued APIs

V2 Name	Descriptive Name	V5 Name
sqlbftsq	Fetch Tablespace Query	sqlbftpq
sqlbstsq	Single Tablespace Query	sqlbstpq
sqlbtsq	Tablespace Query	sqlbmtsq
sqlectdd	Catalog Database	sqlecadb
sqlepstr	Start Database Manager (DB2 Parallel Edition Version 1.2)	sqlepstart
sqlestar	Start Database Manager (DB2 Version 2)	sqlepstart
sqlestop	Stop Database Manager	sqlepstp
sqlubkup	Backup Database	sqlubkp
sqlugrpi	Get Row Partitioning Information (DB2 Parallel Edition Version 1.x)	sqlugrpn
sqluprfw	Rollforward Database (DB2 Parallel Edition Version 1.x)	sqluroll
sqlurllf	Rollforward Database (DB2 Version 2)	sqluroll
sqlursto	Restore Database	sqlurst
sqlxhcom	Commit an Indoubt Transaction	sqlxphcm
sqlxhqry	List Indoubt Transactions	sqlxphqr
sqlxhrol	Roll Back an Indoubt Transaction	sqlxphrl
SQLB-TBSQRY-DATA	Table space data structure.	SQLB-TBSPQRY-DATA
SQLEDBSTRTOPT	Start Database Manager data structure (DB2 Parallel Edition Version 1.2)	SQLE-START-OPTIONS

Appendix H. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, and books. This section describes the information that is provided, and how to access it.

To help you access product information online, DB2 provides the Information Center on OS/2, Windows 95, and the Windows NT operating systems. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. "About the Information Center" on page 536 has more details.

SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available on OS/2, Windows 95, and the Windows NT operating systems. The following table lists the SmartGuides.

SmartGuide	Helps you to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click on **Add**.
Create Database	Create a database, and to perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**New**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, click with the right mouse button on the database you want to tune and select **Configure performance**.
Backup Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to backup and select **Backup**->**Database using SmartGuide**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, click with the right mouse button on the database you want to restore and select **Restore**->**Database using SmartGuide**.
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, click with the right mouse button on the **Tables** icon and select **Create**->**Table using SmartGuide**.
Create Table Space	Create a new table space.	From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create**->**Table space using SmartGuide**.

Online Help

Online help is available with all DB2 components. The following table describes the various types of help.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: **?** *command* where *command* is a keyword or the entire command. For example, **?** *catalog* displays help for all the CATALOG commands, whereas **?** *catalog database* displays help for the CATALOG DATABASE command.
Control Center Help	Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls.	From a window or notebook, click on the **Help** push button or press the F1 key.
Message Help	Describes the cause of a message number, and any action you should take.	From the command line processor in interactive mode, enter: **?** *message number* where *message number* is a valid message number. For example, **?** *SQL30081* displays help about the SQL30081 message. To view message help one screen at a time, enter: **?** *XXXnnnnn* **| more** where *XXX* is the message prefix, such as SQL, and *nnnnn* is the message number, such as 30081. To save message help in a file, enter: **?** *XXXnnnnn* > *filename.ext* where *filename.ext* is the file where you want to save the message help. **Note:** On UNIX-based systems, enter: **\?** *XXXnnnnn* **| more** or **\?** *XXXnnnnn* > *filename.ext*

Type of Help	Contents	How to Access...
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: **help** *statement* where *statement* is an SQL statement. For example, **help** *SELECT* displays help about the SELECT statement.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: **?** *sqlstate* or **?** *class-code* where *sqlstate* is a valid five digit SQL state and *class-code* is a valid two digit class code. For example, **?** *08003* displays help for the 08003 SQL state, whereas **?** *08* displays help for the 08 class code.

DB2 Books

The table in this section lists the DB2 books. They are divided into two groups:

- Cross-platform books: These books are for DB2 on any of the supported platforms.
- Platform-specific books: These books are for DB2 on a specific platform. For example, there is a separate *Quick Beginnings* book for DB2 on OS/2, Windows NT, and UNIX-based operating systems.

Most books are available in HTML and PostScript format, and in hardcopy that you can order from IBM. The exceptions are noted in the table.

You can obtain DB2 books and access information in a variety of different ways:

View To view an HTML book, you can do the following:

- If you are running DB2 administration tools on OS/2, Windows 95, or the Windows NT operating systems, you can use the Information Center. "About the Information Center" on page 536 has more details.
- Use the open file function of the Web browser supplied by DB2 (or one of your own) to open the following page:

  ```
  sqllib/doc/html/index.htm
  ```

  The page contains descriptions of and links to the DB2 books. The path is located on the drive where DB2 is installed.

  You can also open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

Search To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic.
- Click on **Index** at the bottom of any page in an HTML book. Use the Index to find a specific topic in the book.
- Display the Table of Contents or Index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. "About the Information Center" on page 536 has more details.

Print To print a book on a PostScript printer, look for the file name shown in the table.

Order To order a hardcopy book from IBM, use the form number.

Book Name	Book Description	Form Number File Name
	Cross-Platform Books	
Administration Getting Started	Introduces basic DB2 database administration concepts and tasks, and walks you through the primary administrative tasks.	S10J-8154 db2k0x50
Administration Guide	Contains information required to design, implement, and maintain a database to be accessed either locally or in a client/server environment.	S10J-8157 db2d0x50
API Reference	Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications.	S10J-8167 db2b0x50
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	S10J-8159 db2l0x50
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	S10J-8166 db2n0x50
DB2 Connect Enterprise Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Enterprise Edition. Also contains installation and setup information for all supported clients.	S10J-7888 db2cyx50
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Connect Personal Edition.	S10J-8162 db2c1x50
DB2 Connect User's Guide	Provides concepts, programming and general using information about the DB2 Connect products.	S10J-8163 db2c0x50
DB2 Connectivity Supplement	Provides setup and reference information for customers who want to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA Application Requesters with DB2 Universal Database servers, and customers who want to use DRDA Application Servers with DB2 Connect (formerly DDCS) application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x50
Embedded SQL Programming Guide	Explains how to develop applications that access DB2 databases using embedded SQL, and includes discussions about programming techniques and performance considerations.	S10J-8158 db2a0x50
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50

Book Name	Book Description	Form Number File Name
Installing and Configuring DB2 Clients	Provides installation and setup information for all DB2 Client Application Enablers and DB2 Software Developer's Kits. **Note:** Available in HTML and PostScript formats only.	No form number db2iyx50
Master Index	Contains a cross reference to the major topics covered in the DB2 library. **Note:** Available in PostScript format and hardcopy only.	S10J-8170 db2w0x50
Message Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	S10J-8168 db2m0x50
Replication Guide and Reference	Provides planning, configuring, administering, and using information for the IBM Replication tools supplied with DB2.	S95H-0999 db2e0x50
Road Map to DB2 Programming	Introduces the different ways your applications can access DB2, describes key DB2 features you can use in your applications, and points to detailed sources of information for DB2 programming.	S10J-8155 db2u0x50
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	S10J-8156 db2y0x50
SQL Reference	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views.	S10J-8165 db2s0x50
System Monitor Guide and Reference	Describes how to collect different kinds of information about your database and the database manager. Explains how you can use the information to understand database activity, improve performance, and determine the cause of problems.	S10J-8164 db2f0x50
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169 db2p0x50
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database. **Note:** Available in HTML and PostScript formats only.	No form number db2q0x50
Platform-Specific Books		
Building Applications for UNIX Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a UNIX system.	S10J-8161 db2axx50
Building Applications for Windows and OS/2 Environments	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Windows or OS/2 system.	S10J-8160 db2a1x50

Book Name	Book Description	Form Number File Name
DB2 Extended Enterprise Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Extended Enterprise Edition for AIX.	S72H-9620 db2v3x50
DB2 Personal Edition Quick Beginnings	Provides planning, installing, configuring, and using information for DB2 Universal Database Personal Edition on OS/2, Windows 95, and the Windows NT operating systems.	S10J-8150 db2i1x50
DB2 SDK for Macintosh Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Macintosh system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0528 sqla7x02
DB2 SDK for SCO OpenServer Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SCO OpenServer system. **Note:** Available for DB2 Version 2.1.2 only.	S89H-3242 sqla9x02
DB2 SDK for Silicon Graphics IRIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a Silicon Graphics system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S89H-4032 sqlaax02
DB2 SDK for SINIX Building Your Applications	Provides environment setup information and step-by-step instructions to compile, link, and run DB2 applications on a SINIX system. **Note:** Available in PostScript format and hardcopy for DB2 Version 2.1.2 only.	S50H-0530 sqla8x00
Quick Beginnings for OS/2	Provides planning, installing, configuring, and using information for DB2 Universal Database on OS/2. Also contains installing and setup information for all supported clients.	S10J-8147 db2i2x50
Quick Beginnings for UNIX	Provides planning, installing, configuring, and using information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for all supported clients.	S10J-8148 db2ixx50
Quick Beginnings for Windows NT	Provides planning, installing, configuring, and using information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for all supported clients.	S10J-8149 db2i6x50

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e50` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

Language	Identifier	Language	Identifier
Brazilian Portuguese	B	Hungarian	H
Bulgarian	U	Italian	I
Czech	X	Norwegian	N
Danish	D	Polish	P
English	E	Russian	R
Finnish	Y	Slovenian	L
French	F	Spanish	Z
German	G	Swedish	S

2. For late breaking information that could not be included in the DB2 books, see the README file. Each DB2 product includes a README file which you can find in the directory where the product is installed.

About the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on OS/2, Windows 95, and the Windows NT operating systems. You must install the DB2 administration tools to see the Information Center.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu.

The Information Center provides the following kinds of information. Click on the appropriate tab to look at the information:

Tasks	Lists tasks you can perform using DB2.
Reference	Lists DB2 reference information, such as keywords, commands, and APIs.
Books	Lists DB2 books.
Troubleshooting	Lists categories of error messages and their recovery actions.
Sample Programs	Lists sample programs that come with the DB2 Software Developer's Kit. If the Software Developer's Kit is not installed, this tab is not displayed.
Web	Lists DB2 information on the World Wide Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides search capabilities so you can look for specific topics, and filter capabilities to limit the scope of your searches.

Appendix I. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

IBM Director of Licensing,
IBM Corporation,
500 Columbus Avenue,
Thornwood, NY, 10594
USA.

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 which has been exchanged, should contact:

IBM Canada Limited
Department 071
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication 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.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

ACF/VTAM	MVS/ESA
ADSTAR	MVS/XA
AISPO	NetView
AIX	OS/400
AIXwindows	OS/390
AnyNet	OS/2
APPN	PowerPC
AS/400	QMF
CICS	RACF
C Set++	RISC System/6000
C/370	SAA
DATABASE 2	SP
DatagLANce	SQL/DS
DataHub	SQL/400
DataJoiner	S/370
DataPropagator	System/370
DataRefresher	System/390
DB2	SystemView
Distributed Relational Database Architecture	VisualAge
DRDA	VM/ESA
Extended Services	VSE/ESA
FFST	VTAM
First Failure Support Technology	WIN-OS/2
IBM	
IMS	
Lan Distance	

Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

C-bus is a trademark of Corollary, Inc.

HP-UX is a trademark of Hewlett-Packard.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Solaris is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Index

Contacting IBM

This section lists ways you can get more information from IBM.

If you have a technical problem, please take the time to review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. Depending on the nature of your problem or concern, this guide will suggest information you can gather to help us to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

Telephone

If you live in the U.S.A., call one of the following numbers:

- 1-800-237-5511 to learn about available service options.
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, see Appendix A of the IBM Software Support Handbook. You can access this document by selecting the "Roadmap to IBM Support" item at: http://www.ibm.com/support/.

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

World Wide Web
http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

Anonymous FTP Sites
ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

Internet Newsgroups
comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe
GO IBMDB2 to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

Part Number: 10J8167

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

10J8167

S10J-8167-00