

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows



Call Level Interface Guide and Reference, Volume 1

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows



Call Level Interface Guide and Reference, Volume 1

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

Edition Notice

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

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

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

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

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

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

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

DB2 V9 BETA

Contents

Part 1. CLI background information 1

Chapter 1. Introduction to CLI 3

Introduction to DB2 CLI and ODBC	3
DB2 Call Level Interface (CLI) versus embedded SQL	4
DB2 CLI and ODBC drivers	7

Chapter 2. DB2 CLI and ODBC 9

Comparison of DB2 CLI and Microsoft ODBC	9
--	---

Part 2. Programming CLI applications 13

Chapter 3. Writing a basic CLI application 15

Initialization	15
Handles in CLI	15
Initialization and termination in CLI overview.	17
Initializing CLI applications	18
Transaction processing.	20
Transaction processing in CLI overview	20
Allocating statement handles in CLI applications	22
Issuing SQL statements in CLI applications.	23
Preparing and executing SQL statements in CLI applications	24
Deferred prepare in CLI applications	26
Parameter marker binding in CLI applications.	26
Binding parameter markers in CLI applications	29
Commit modes in CLI applications	30
When to call the CLI SQLEndTran() function	32
Retrieving query results in CLI applications	33
Updating and deleting data in CLI applications	35
Freeing statement resources in CLI applications	37
Handle freeing in CLI applications	38
Data types and data conversion in CLI applications	39
SQL symbolic and default data types for CLI applications	41
C data types for CLI applications	43
String handling in CLI applications	47
Diagnostics in CLI applications overview	49
CLI function return codes	50
SQLSTATES for DB2 CLI	51
Termination	53
Terminating a CLI application	53

Chapter 4. Programming hints and tips 55

Programming hints and tips for CLI applications.	55
Reduction of network flows with CLI array input chaining	62

Chapter 5. Cursors 65

Cursors.	65
------------------	----

Cursors in CLI applications	65
Cursor considerations for CLI applications	68
Result sets.	70
Result set terminology in CLI applications	70
Rowset retrieval examples in CLI applications.	71
Specifying the rowset returned from the result set	73
Retrieving data with scrollable cursors in a CLI application	76
Bookmarks	78
Bookmarks in CLI applications	78
Retrieving data with bookmarks in a CLI application	79

Chapter 6. Array input and output . . . 81

Array input	81
Binding parameter markers in CLI applications with column-wise array input	81
Binding parameter markers in CLI applications with row-wise array input	82
Parameter diagnostic information in CLI applications	83
Changing parameter bindings in CLI applications with offsets	84
Array output	85
Column binding in CLI applications	85
Result set retrieval into arrays in CLI applications	87
Retrieving array data in CLI applications using column-wise binding	89
Retrieving array data in CLI applications using row-wise binding	90
Changing column bindings in a CLI application with column binding offsets.	91

Chapter 7. Working with large amounts of data 93

Specifying parameter values at execute time for long data manipulation in CLI applications	93
Data retrieval in pieces in CLI applications.	95
Large object usage in CLI applications	96
LOB locators in CLI applications	98
Fetching LOB data with LOB locators in CLI applications	99
Direct file input and output for LOB handling in CLI applications	101
LOB usage in ODBC applications.	102
Bulk data manipulation	103
Long data for bulk inserts and updates in CLI applications	103
Retrieving bulk data with bookmarks using SQLBulkOperations() in CLI applications	105
Inserting bulk data with bookmarks using SQLBulkOperations() in CLI applications	106
Updating bulk data with bookmarks using SQLBulkOperations() in CLI applications	107

Deleting bulk data with bookmarks using SQLBulkOperations() in CLI applications	109
Importing data with the CLI LOAD utility in CLI applications	110
Chapter 8. Stored procedures	113
Calling stored procedures from CLI applications	113
DB2 CLI stored procedure commit behavior	115
Chapter 9. XML data	117
XML data handling in CLI applications - Overview	117
XML column inserts and updates in CLI applications	118
XML data retrieval in CLI applications	119
Changing of default XML type handling in CLI applications	120
Chapter 10. Compound SQL	123
Executing compound SQL statements in CLI applications	123
Return codes for compound SQL in CLI applications	125
Chapter 11. Asynchronous execution of CLI functions	127
Asynchronous execution of CLI functions	127
Executing functions asynchronously in CLI applications	129
Chapter 12. Multithreaded CLI applications	133
Multithreaded CLI applications	133
Application model for multithreaded CLI applications	134
Mixed multithreaded CLI applications	136
Chapter 13. Multisite updates (two phase commit).	137
Multisite updates (two phase commit) in CLI applications	137
DB2 as transaction manager in CLI applications	138
Process-based XA-compliant Transaction Program Monitor (XA TP) programming considerations for CLI applications	141
Chapter 14. Unicode	143
Unicode CLI applications	143
Unicode functions (CLI)	144
Unicode function calls to ODBC driver managers	145
Chapter 15. User-defined types (UDT)	147
Distinct type usage in CLI applications	147
User-defined type (UDT) usage in CLI applications	148
Chapter 16. Descriptors	151
Descriptors in CLI applications	151
Consistency checks for descriptors in CLI applications	154

Descriptor allocation and freeing	155
Descriptor manipulation with descriptor handles in CLI applications	158
Descriptor manipulation without using descriptor handles in CLI applications.	160
Chapter 17. Environment, connection, and statement attributes	163
Environment, connection, and statement attributes in CLI applications	163
Chapter 18. Querying system catalog information	167
Catalog functions for querying system catalog information in CLI applications	167
Input arguments on catalog functions in CLI applications	168
Chapter 19. Vendor escape clauses	171
Vendor escape clauses in CLI applications	171
Extended scalar functions for CLI applications	174
Chapter 20. Mixing embedded SQL and DB2 CLI.	183
Considerations for mixing embedded SQL and DB2 CLI.	183
Chapter 21. CLI/ODBC/JDBC Static Profiling	185
Creating static SQL with CLI/ODBC/JDBC Static Profiling	185
Capture file for CLI/ODBC/JDBC Static Profiling	187
Chapter 22. CLI/ODBC/JDBC trace facility	189
CLI/ODBC/JDBC trace facility	189
CLI and JDBC trace files.	194
Chapter 23. CLI bind files and package names	203
DB2 CLI bind files and package names	203
Bind option limitations for CLI packages	205
<hr/>	
Part 3. CLI environment and application building	207
Chapter 24. CLI environmental setup	209
Setting up the CLI environment	209
Setting up the UNIX ODBC environment	210
Setting up the unixODBC Driver Manager.	212
Sample build scripts and configurations for the unixODBC Driver Manager.	213
Setting up the Windows CLI environment.	215
Selecting a different DB2 copy for your Windows CLI application.	217
Chapter 25. Building CLI applications	219

UNIX	219
Building CLI applications on UNIX	219
Building CLI multi-connection applications on UNIX	220
Building CLI routines on UNIX	222
AIX	224
HP-UX	227
Linux	227
Solaris	228
Windows	228
Building CLI applications on Windows	228
Building CLI multi-connection applications on Windows	229
Building CLI routines on Windows	231
Batch file for Windows applications	233
Batch file for Windows routines	233

Chapter 26. CLI sample programs	235
CLI sample programs	235
CLI samples	235

Part 4. IBM DB2 Driver for ODBC and CLI 239

Chapter 27. IBM DB2 Driver for ODBC and CLI overview.	241
--	------------

Chapter 28. Obtaining the IBM DB2 Driver for ODBC and CLI	243
Obtaining the IBM DB2 Driver for ODBC and CLI	243

Chapter 29. Installing the IBM DB2 Driver for ODBC and CLI	245
Installing the IBM DB2 Driver for ODBC and CLI	245
Installing multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine	246
Intalling the IBM DB2 Driver for ODBC and CLI on a machine with an existing DB2 client	247

Chapter 30. Configuring the IBM DB2 Driver for ODBC and CLI	249
Configuring the IBM DB2 Driver for ODBC and CLI.	249
Configuring environment variables for the IBM DB2 Driver for ODBC and CLI	250
Configuring environment variables for the IBM DB2 Driver for ODBC and CLI	250
Environment variables supported by the IBM DB2 Driver for ODBC and CLI	251
db2oregl.exe overview	252
Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft DTC	253
Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager	254

Chapter 31. Connecting to databases with the IBM DB2 Driver for ODBC and CLI	257
---	------------

Connecting to databases with the IBM DB2 Driver for ODBC and CLI	257
Registering ODBC data sources for applications using the IBM DB2 Driver for ODBC and CLI	259
Using security plugins with the IBM DB2 Driver for ODBC and CLI	261

Chapter 32. Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI 263

Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI	263
DB2 CLI and ODBC API support in the IBM DB2 Driver for ODBC and CLI	264
XA API support in the IBM DB2 Driver for ODBC and CLI	265
LDAP support in the IBM DB2 Driver for ODBC and CLI	265
Restrictions of the IBM DB2 Driver for ODBC and CLI.	266
Diagnostic support in the IBM DB2 Driver for ODBC and CLI.	266

Chapter 33. Deploying the IBM DB2 Driver for ODBC and CLI with database applications. 269

Deploying the IBM DB2 Driver for ODBC and CLI with database applications	269
License requirements for the IBM DB2 Driver for ODBC and CLI.	270

Chapter 34. Uninstalling the IBM DB2 Driver for ODBC and CLI 273

Uninstalling the IBM DB2 Driver for ODBC and CLI.	273
---	-----

Part 5. CLI/ODBC configuration keywords 275

Chapter 35. CLI/ODBC configuration keywords 277

db2cli.ini initialization file	277
CLI/ODBC configuration keywords listing by category	279
AllowGetDataLOBReaccess CLI/ODBC configuration keyword	284
AltHostName CLI/ODBC configuration keyword	284
AltPort CLI/ODBC configuration keyword	285
AppendAPIName CLI/ODBC configuration keyword	285
AppendRowColToErrorMessage CLI/ODBC configuration keyword	286
AppendForFetchOnly CLI/ODBC configuration keyword	287
AppUsesLOBLocator CLI/ODBC configuration keyword	287
ArrayInputChain CLI/ODBC configuration keyword	288

AsyncEnable CLI/ODBC configuration keyword	289	DisableKeysetCursor CLI/ODBC configuration keyword	321
Authentication CLI/ODBC configuration keyword	290	DisableMultiThread CLI/ODBC configuration keyword	322
AutoCommit CLI/ODBC configuration keyword	290	DisableUnicode CLI/ODBC configuration keyword	322
BIDI CLI/ODBC configuration keyword	291	DSN CLI/ODBC configuration keyword	323
BitData CLI/ODBC configuration keyword	292	FileDSN CLI/ODBC configuration keyword	323
BlockForNRows CLI/ODBC configuration keyword	292	FloatPrecRadix CLI/ODBC configuration keyword	324
BlockLobs CLI/ODBC configuration keyword	293	GranteeList CLI/ODBC configuration keyword	324
CheckForFork CLI/ODBC configuration keyword	294	GrantorList CLI/ODBC configuration keyword	325
ClientAcctStr CLI/ODBC configuration keyword	294	Graphic CLI/ODBC configuration keyword	326
ClientAppName CLI/ODBC configuration keyword	295	Hostname CLI/ODBC configuration keyword	327
ClientBuffersUnboundLOBS CLI/ODBC configuration keyword	296	IgnoreWarnings CLI/ODBC configuration keyword	327
ClientUserID CLI/ODBC configuration keyword	297	IgnoreWarnList CLI/ODBC configuration keyword	328
ClientWrkStnName CLI/ODBC configuration keyword	298	Instance CLI/ODBC configuration keyword	328
CLIPkg CLI/ODBC configuration keyword	299	Interrupt CLI/ODBC configuration keyword	329
ConnectNode CLI/ODBC configuration keyword	299	KeepDynamic CLI/ODBC configuration keyword	329
ConnectTimeout CLI/ODBC configuration keyword	300	KRBPlugin CLI/ODBC configuration keyword	330
ConnectType CLI/ODBC configuration keyword	301	LoadXAInterceptor CLI/ODBC configuration keyword	330
CurrentFunctionPath CLI/ODBC configuration keyword	302	LOBCacheSize CLI/ODBC configuration keyword	331
CurrentImplicitXMLParseOption CLI/ODBC configuration keyword	302	LOBFileThreshold CLI/ODBC configuration keyword	332
CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword	303	LOBMaxColumnSize CLI/ODBC configuration keyword	332
CurrentPackagePath CLI/ODBC configuration keyword	304	LockTimeout CLI/ODBC configuration keyword	333
CurrentPackageSet CLI/ODBC configuration keyword	305	LongDataCompat CLI/ODBC configuration keyword	333
CurrentRefreshAge CLI/ODBC configuration keyword	306	MapBigintCDefault CLI/ODBC configuration keyword	334
CurrentSchema CLI/ODBC configuration keyword	306	MapCharToWChar CLI/ODBC configuration keyword	335
CurrentSQLID CLI/ODBC configuration keyword	306	MapDateCDefault CLI/ODBC configuration keyword	336
CursorHold CLI/ODBC configuration keyword	307	MapDateDescribe CLI/ODBC configuration keyword	337
CursorTypes CLI/ODBC configuration keyword	308	MapDecimalFloatDescribe CLI/ODBC configuration keyword	338
Database CLI/ODBC configuration keyword	309	MapGraphicDescribe CLI/ODBC configuration keyword	339
DateTimeStringFormat CLI/ODBC configuration keyword	310	MapTimeCDefault CLI/ODBC configuration keyword	340
DB2Degree CLI/ODBC configuration keyword	311	MapTimeDescribe CLI/ODBC configuration keyword	340
DB2Explain CLI/ODBC configuration keyword	311	MapTimestampCDefault CLI/ODBC configuration keyword	342
DB2NETNamedParam CLI/ODBC configuration keyword	312	MapTimestampDescribe CLI/ODBC configuration keyword	342
DB2Optimization CLI/ODBC configuration keyword	313	MapXMLCDefault CLI/ODBC configuration keyword	343
DBAlias CLI/ODBC configuration keyword	313	MapXMLDescribe CLI/ODBC configuration keyword	344
DBName CLI/ODBC configuration keyword	314	MaxLOBBlockSize CLI/ODBC configuration keyword	345
DecimalFloatRoundingMode CLI/ODBC configuration keyword	315	Mode CLI/ODBC configuration keyword	346
DeferredPrepare CLI/ODBC configuration keyword	316	NotifyLevel CLI/ODBC configuration keyword	346
DescribeCall CLI/ODBC configuration keyword	317	OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword	347
DescribeInputOnPrepare CLI/ODBC configuration keyword	318	OleDbReturnCharAsWChar CLI/ODBC configuration keyword	348
DescribeOutputLevel CLI/ODBC configuration keyword	318		
DescribeParam CLI/ODBC configuration keyword	320		
DiagLevel CLI/ODBC configuration keyword	321		
DiagPath CLI/ODBC configuration keyword	321		

OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword	348
OptimizeForNRows CLI/ODBC configuration keyword	349
Patch1 CLI/ODBC configuration keyword.	350
Patch2 CLI/ODBC configuration keyword.	352
Port CLI/ODBC configuration keyword	355
ProgramName CLI/ODBC configuration keyword	356
Protocol CLI/ODBC configuration keyword	356
PWD CLI/ODBC configuration keyword	357
PWDPlugin CLI/ODBC configuration keyword	358
QueryTimeoutInterval CLI/ODBC configuration keyword	358
ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword	359
ReceiveTimeout CLI/ODBC configuration keyword	360
Reopt CLI/ODBC configuration keyword	360
ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword	361
ReportPublicPrivileges CLI/ODBC configuration keyword	362
RetCatalogAsCurrServer CLI/ODBC configuration keyword	362
RetOleDbConnStr CLI/ODBC configuration keyword	363
RetryOnError CLI/ODBC configuration keyword	364
SaveFile CLI/ODBC configuration keyword	365
SchemaList CLI/ODBC configuration keyword	365
ServerMsgMask CLI/ODBC configuration keyword	366
ServiceName CLI/ODBC configuration keyword	366
SkipTrace CLI/ODBC configuration keyword.	367
SQLOverrideFileName CLI/ODBC configuration keyword	368
StaticCapFile CLI/ODBC configuration keyword	368
StaticLogFile CLI/ODBC configuration keyword	369
StaticMode CLI/ODBC configuration keyword	370
StaticPackage CLI/ODBC configuration keyword	370
StreamGetData CLI/ODBC configuration keyword	371
StreamPutData CLI/ODBC configuration keyword	372
SysSchema CLI/ODBC Configuration Keyword	372
TableType CLI/ODBC configuration keyword	374
TempDir CLI/ODBC configuration keyword	374
Trace CLI/ODBC configuration keyword	375
TraceComm CLI/ODBC configuration keyword	376
TraceErrImmediate CLI/ODBC configuration keyword	377
TraceFileName CLI/ODBC configuration keyword	378
TraceFlush CLI/ODBC configuration keyword	379
TraceFlushOnError CLI/ODBC configuration keyword	379
TraceLocks CLI/ODBC configuration keyword	380
TracePathName CLI/ODBC configuration keyword	381
TracePIDList CLI/ODBC configuration keyword	382
TracePIDENTID CLI/ODBC configuration keyword	383
TraceRefreshInterval CLI/ODBC configuration keyword	383
TraceStmtOnly CLI/ODBC configuration keyword	384

TraceTime CLI/ODBC configuration keyword	385
TraceTimestamp CLI/ODBC configuration keyword	386
Trusted_Connection CLI/ODBC configuration keyword	386
TxnIsolation CLI/ODBC configuration keyword	388
UID CLI/ODBC configuration keyword	388
Underscore CLI/ODBC configuration keyword	389
UseOldStpCall CLI/ODBC configuration keyword	390
UseServerMsgSP CLI/ODBC configuration keyword	391
WarningList CLI/ODBC configuration keyword	391
XMLDeclaration CLI/ODBC configuration keyword	392

Part 6. Data conversion 395

Chapter 36. Data conversion 397

Data conversions supported in CLI	397
SQL to C data conversion in CLI	399
C to SQL data conversion in CLI	406

Part 7. Appendixes 413

Appendix A. DB2 Database technical information 415

Overview of the DB2 technical information	415
Documentation feedback	415
DB2 technical library in PDF format.	416
Ordering printed DB2 books	418
Displaying SQL state help from the command line processor.	419
Accessing different versions of the DB2 Information Center	419
Displaying topics in your preferred language in the DB2 Information Center.	420
Updating the DB2 Information Center installed on your computer or intranet server.	421
DB2 Visual Explain tutorial.	422
DB2 troubleshooting information.	423
Terms and Conditions	423

Appendix B. Notices 425

Trademarks	427
----------------------	-----

Appendix C. Further notices for the DB2 Call Level Interface Guide and Reference 429

Index 431

Contacting IBM 443

Part 1. CLI background information

Chapter 1. Introduction to CLI

Introduction to DB2 CLI and ODBC	3	DB2 CLI and ODBC drivers	7
DB2 Call Level Interface (CLI) versus embedded SQL	4		

Introduction to DB2 CLI and ODBC

DB2 Call Level Interface (DB2[®] CLI) is IBM's callable SQL interface to the DB2 family of database servers. It is a 'C' and 'C++' application programming interface for relational database access that uses function calls to pass dynamic SQL statements as function arguments. It is an alternative to embedded dynamic SQL, but unlike embedded SQL, DB2 CLI does not require host variables or a precompiler.

DB2 CLI is based on the Microsoft[®]** Open Database Connectivity** (ODBC) specification, and the International Standard for SQL/CLI. These specifications were chosen as the basis for the DB2 Call Level Interface in an effort to follow industry standards and to provide a shorter learning curve for those application programmers already familiar with either of these database interfaces. In addition, some DB2 specific extensions have been added to help the application programmer specifically exploit DB2 features.

The DB2 CLI driver also acts as an ODBC driver when loaded by an ODBC driver manager. It conforms to ODBC 3.51.

DB2 CLI Background information:

To understand DB2 CLI or any callable SQL interface, it is helpful to understand what it is based on, and to compare it with existing interfaces.

The X/Open Company and the SQL Access Group jointly developed a specification for a callable SQL interface referred to as the *X/Open Call Level Interface*. The goal of this interface is to increase the portability of applications by enabling them to become independent of any one database vendor's programming interface. Most of the X/Open Call Level Interface specification has been accepted as part of the ISO Call Level Interface International Standard (ISO/IEC 9075-3:1995 SQL/CLI).

Microsoft developed a callable SQL interface called Open Database Connectivity (ODBC) for Microsoft operating systems based on a preliminary draft of X/Open CLI.

The ODBC specification also includes an operating environment where database specific ODBC Drivers are dynamically loaded at run time by a driver manager based on the data source (database name) provided on the connect request. The application is linked directly to a single driver manager library rather than to each DBMS's library. The driver manager mediates the application's function calls at run time and ensures they are directed to the appropriate DBMS specific ODBC driver. Since the ODBC driver manager only knows about the ODBC-specific functions, DBMS-specific functions cannot be accessed in an ODBC environment. DBMS-specific dynamic SQL statements are supported via a mechanism called an escape clause.

ODBC is not limited to Microsoft operating systems; other implementations are available on various platforms.

The DB2 CLI load library can be loaded as an ODBC driver by an ODBC driver manager. For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows® platform, the ODBC SDK is available as part of the Microsoft Data Access Components (MDAC) SDK, available for download from <http://www.microsoft.com/data/>. For non-Windows platforms, the ODBC SDK is provided by other vendors. When developing ODBC applications that may connect to DB2 servers, use the Call Level Interface Guide and Reference, Volume 1 and the Call Level Interface Guide and Reference, Volume 2 (for information on DB2 specific extensions and diagnostic information), in conjunction with the ODBC Programmer's Reference and SDK Guide available from Microsoft.

Applications written directly to DB2 CLI link directly to the DB2 CLI load library. DB2 CLI includes support for many ODBC and ISO SQL/CLI functions, as well as DB2 specific functions.

The following DB2 features are available to both ODBC and DB2 CLI applications:

- double byte (graphic) data types
- stored procedures
- Distributed Unit of Work (DUOW), two phase commit
- compound SQL
- user defined types (UDT)
- user defined functions (UDF)

Related concepts:

- "Comparison of DB2 CLI and Microsoft ODBC" on page 9
- "DB2 Call Level Interface (CLI) versus embedded SQL" on page 4

DB2 Call Level Interface (CLI) versus embedded SQL

Two important differences between embedded SQL and DB2 CLI are:

1. the method for building applications;

To build an embedded application:

- a. connect to the target database(s) and precompile the embedded SQL;
- b. compile the application code; and
- c. bind the application to the target database(s).

After this, you can run the application against the target database(s).

To build a CLI application:

- a. compile the application code.

After this, you can run the application against any DB2 database to which the bind files shipped with CLI have been bound.

Because of this difference, deploying CLI applications is easier than deploying embedded SQL applications, particularly in environments where the target databases are not known at build time or where the target databases might change.

and

2. it is much easier to use static SQL in embedded SQL applications than in DB2 CLI applications.

For this reason, whether you use static SQL will be a factor in choosing whether to use embedded SQL or DB2 CLI.

Strengths and weaknesses of embedded SQL and DB2 CLI:

When deciding whether to create your applications using DB2 CLI or embedded SQL, consider the following strengths and weaknesses of DB2 CLI and embedded SQL:

- DB2 CLI applications are easier to deploy and compatible across database management systems:
 - Embedded SQL applications must be precompiled, and are therefore dependant on precompiler software. DB2 CLI applications do not need to be precompiled.
 - You must bind embedded SQL applications to target databases, but you do not need to bind CLI applications to target databases. There are bind files shipped with DB2 CLI that need to be bound to target databases once for all CLI applications.
 - CLI provides a consistent interface to query catalog information contained in DBMS catalog tables; and the result sets returned are consistent across DBMSs. There is no such consistent interface available with embedded SQL.
- DB2 CLI is more flexible at runtime:
 - You can use parameter markers in CLI applications, but not in embedded SQL applications. CLI applications can process statements with parameter markers, and obtain data type information for those parameters using describe information.
 - CLI provides enhanced parameter input and fetching capability, allowing arrays of data to be specified on input, retrieving multiple rows of a result set directly into an array, and executing statements that generate multiple result sets.
- Some functionality is easier to achieve using DB2 CLI than using embedded SQL:
 - CLI objects like connection handles allow you to easily connect to multiple databases, and have multiple connections to the same database in a CLI application. To do the same thing in an embedded SQL application, you must use multi-threading, which entails using DB2 context management APIs to be thread-safe.

See: Mixed multithreaded CLI applications
 - Cursor management can be easier with DB2 CLI than with embedded SQL. For example, DB2 CLI does not require the explicit declaration of cursors, it has a pool of cursors that get used as needed. Also, you do not need to open cursors in CLI applications, the execution of a SELECT statement automatically causes a cursor to be opened. Finally, CLI has more extensive support for scrollable cursors, including using scrollable cursors with array output.
 - Embedded SQL applications use complex data areas, such as SQLDA and SQLCA. DB2 CLI provides handles to reference the same kind of information. For example, environment handles, connection handles, statement handles, and descriptor handles provide a means to refer to information like global variables, connection specific information, the parameters of an SQL statement, or the columns of a result set.

- It is easier to convert between various SQL and C data types in DB2 CLI applications than in embedded SQL applications, because DB2 CLI provides extended data conversion support that embedded SQL does not.
- Some functionality is easier to achieve using embedded SQL than using DB2 CLI:
 - Embedded SQL is supported in languages other than C or C++.
 - It is much easier to use static SQL in embedded SQL applications.

Strengths and weaknesses of static SQL and dynamic SQL:

When choosing whether to use static SQL, consider the following:

- Static SQL can perform better than dynamic SQL:

In CLI applications, dynamic SQL is prepared at run time. In embedded SQL applications, static SQL is prepared at precompile time. The performance cost of preparing statements at runtime is: 1) processing the prepare (precompiling the SQL); and 2) sending the prepare across the network.

It is important to note that despite these costs, static SQL will not always have better performance than dynamic SQL. For example, you can avoid precompiling dynamic SQL statements by caching the statements. Also, you can avoid the additional network traffic of preparing dynamic SQL by using deferred prepare. Finally, database statistics available when preparing dynamic SQL at runtime may result in a better access plan than database statistics available when preparing static SQL at BIND time. For example, there may be new indexes at runtime. A better access plan would result in better performance.
- Implementing security rules can be easier with static SQL than with dynamic SQL:

With static SQL, authorization to access objects like tables and views is associated with a package and is validated at package binding time. This means that a database administrator can grant execute access for a set of users on a particular package without having to grant those users explicit access to each database object. With dynamic SQL, authorization is validated at run time on a per statement basis. This means an administrator must grant explicit access to each database object, which permits these users access to parts of the objects that they do not have a need to access.

Static SQL in a DB2 CLI application:

It is possible to use static SQL in DB2 CLI applications:

- Using stored procedures with static SQL

It is possible to make use of static SQL within a CLI application by creating a stored procedure that contains the static SQL. The stored procedure is called from within a CLI application and is executed on the server. Once the stored procedure is created, any CLI or ODBC application can call it.
- Mixing embedded SQL and DB2 CLI

It is also possible to write a mixed application that uses both embedded SQL and DB2 CLI. In this case, CLI is used to provide the base application, with key modules written in embedded SQL using static SQL for performance or security reasons. This complicates the application design, and should only be used if stored procedures do not meet the application requirements.

See: Considerations for mixing embedded SQL and DB2 CLI and `dbusemx.sqc` -- How to execute embedded SQL statements in CLI

Related concepts:

- “Choosing an application programming interface” in *Getting Started with Database Application Development*
- “Supported database application programming interfaces” in *Getting Started with Database Application Development*

DB2 CLI and ODBC drivers

In the DB2 Client and the DB2 Runtime Client there is a driver for the DB2 CLI application programming interface (API) and the ODBC API. This driver is commonly referred to throughout the DB2 Information Center and DB2 books as the DB2 CLI driver or the DB2 CLI/ODBC driver.

New with DB2 Version 9, there is also a separate CLI and ODBC driver called the IBM® DB2 Driver for ODBC and CLI. The IBM DB2 Driver for ODBC and CLI provides runtime support for the DB2 CLI and ODBC APIs. However, this driver is installed and configured separately, and supports a subset of the functionality of the DB2 clients, such as connectivity, in addition to the CLI and ODBC API support.

Information that applies to the CLI and ODBC driver that is part of the DB2 client generally applies to the IBM DB2 Driver for ODBC and CLI too. However, there are some restrictions and some functionality that is unique to the IBM DB2 Driver for ODBC and CLI. Information that applies only to the IBM DB2 Driver for ODBC and CLI will use the full title of the driver to distinguish it from general information that applies to the ODBC and CLI driver that comes with the DB2 clients.

For more information about the IBM DB2 Driver for ODBC and CLI, see: IBM DB2 Driver for ODBC and CLI overview.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241
- “Introduction to DB2 CLI and ODBC” on page 3

Chapter 2. DB2 CLI and ODBC

Comparison of DB2 CLI and Microsoft ODBC

This topic discusses the support provided by the DB2 ODBC driver, and how it differs from DB2 CLI.

Figure 1 below compares DB2 CLI and the DB2 ODBC driver. The left side shows an ODBC driver under the ODBC Driver Manager, and the right side illustrates DB2 CLI, the callable interface designed for DB2 specific applications.

DB2 Client refers to all available DB2 Clients. DB2 refers to all DB2 Database for Linux®, UNIX®, and Windows products.

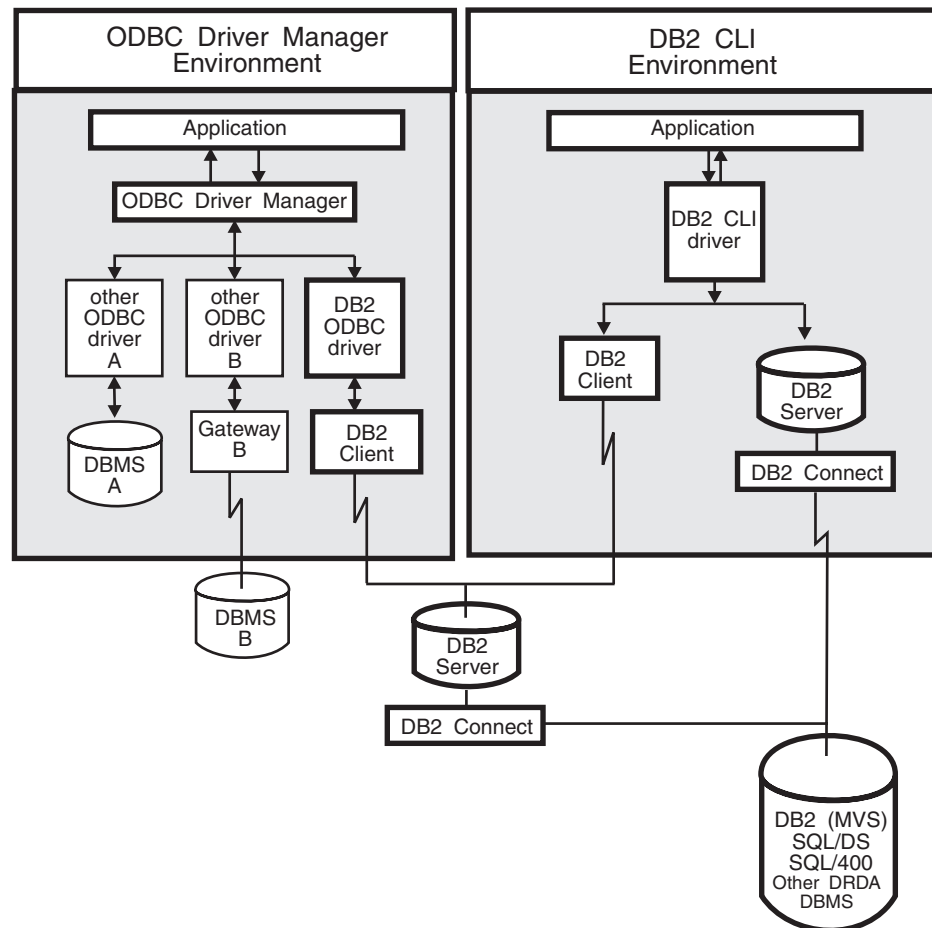


Figure 1. DB2 CLI and ODBC.

In an ODBC environment, the Driver Manager provides the interface to the application. It also dynamically loads the necessary *driver* for the database server that the application connects to. It is the driver that implements the ODBC function set, with the exception of some extended functions implemented by the Driver Manager. In this environment DB2 CLI conforms to ODBC 3.51.

For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows platform, the ODBC SDK is available as part of the Microsoft Data Access Components (MDAC) SDK, available for download from <http://www.microsoft.com/data/>. For non-Windows platforms, the ODBC SDK is provided by other vendors.

In environments without an ODBC driver manager, DB2 CLI is a self sufficient driver which supports a subset of the functions provided by the ODBC driver. Table 1 summarizes the two levels of support, and the CLI and ODBC function summary provides a complete list of ODBC functions and indicates if they are supported.

Table 1. DB2 CLI ODBC support

ODBC features	DB2 ODBC Driver	DB2 CLI
Core level functions	All	All
Level 1 functions	All	All
Level 2 functions	All	All, except for SQLDrivers()
Additional DB2 CLI functions	All, functions can be accessed by dynamically loading the DB2 CLI library.	<ul style="list-style-type: none"> • SQLSetConnectAttr() • SQLGetEnvAttr() • SQLSetEnvAttr() • SQLSetColAttributes() • SQLGetSQLCA() • SQLBindFileToCol() • SQLBindFileToParam() • SQLExtendedBind() • SQLExtendedPrepare() • SQLGetLength() • SQLGetPosition() • SQLGetSubString()

Table 1. DB2 CLI ODBC support (continued)

ODBC features	DB2 ODBC Driver	DB2 CLI
SQL data types	All the types listed for DB2 CLI.	<ul style="list-style-type: none"> • SQL_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONG • SQL_LONGVARIABLE • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SHORT • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR
C data types	All the types listed for DB2 CLI.	<ul style="list-style-type: none"> • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DATE • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TIME • SQL_C_TIMESTAMP • SQL_C_TINYINT • SQL_C_SBIGINT • SQL_C_UBIGINT • SQL_C_NUMERIC ** • SQL_C_WCHAR <p>** Only supported on Windows platform</p>

Table 1. DB2 CLI ODBC support (continued)

ODBC features	DB2 ODBC Driver	DB2 CLI
Return codes	All the codes listed for DB2 CLI.	<ul style="list-style-type: none"> • SQL_SUCCESS • SQL_SUCCESS_WITH_INFO • SQL_STILL_EXECUTING • SQL_NEED_DATA • SQL_NO_DATA_FOUND • SQL_ERROR • SQL_INVALID_HANDLE
SQLSTATES	Mapped to X/Open SQLSTATES with additional IBM SQLSTATES, with the exception of the ODBC type 08S01.	Mapped to X/Open SQLSTATES with additional IBM SQLSTATES
Multiple connections per application	Supported	Supported
Dynamic loading of driver	Supported	Not applicable

Isolation levels:

The following table map IBM RDBMs isolation levels to ODBC transaction isolation levels. The SQLGetInfo() function indicates which isolation levels are available.

Table 2. Isolation levels under ODBC

IBM isolation level	ODBC isolation level
Cursor stability	SQL_TXN_READ_COMMITTED
Repeatable read	SQL_TXN_SERIALIZABLE_READ
Read stability	SQL_TXN_REPEATABLE_READ
Uncommitted read	SQL_TXN_READ_UNCOMMITTED
No commit	(no equivalent in ODBC)
Note: SQLSetConnectAttr() and SQLSetStmtAttr() will return SQL_ERROR with an SQLSTATE of HY009 if you try to set an unsupported isolation level.	

Restriction:

Mixing ODBC and DB2 CLI features and function calls in an application is not supported on the Windows 64-bit operating system.

Related concepts:

- “Introduction to DB2 CLI and ODBC” on page 3
- “Isolation levels” in *SQL Reference, Volume 1*

Related reference:

- “CLI function return codes” on page 50
- “SQL symbolic and default data types for CLI applications” on page 41
- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*

Part 2. Programming CLI applications

Chapter 3. Writing a basic CLI application

Initialization	15	Retrieving query results in CLI applications	33
Handles in CLI	15	Updating and deleting data in CLI applications	35
Initialization and termination in CLI overview.	17	Freeing statement resources in CLI applications	37
Initializing CLI applications	18	Handle freeing in CLI applications	38
Transaction processing.	20	Data types and data conversion in CLI applications	39
Transaction processing in CLI overview	20	SQL symbolic and default data types for CLI applications	41
Allocating statement handles in CLI applications	22	C data types for CLI applications	43
Issuing SQL statements in CLI applications.	23	String handling in CLI applications	47
Preparing and executing SQL statements in CLI applications	24	Diagnostics in CLI applications overview	49
Deferred prepare in CLI applications	26	CLI function return codes	50
Parameter marker binding in CLI applications.	26	SQLSTATES for DB2 CLI	51
Binding parameter markers in CLI applications	29	Termination	53
Commit modes in CLI applications	30	Terminating a CLI application	53
When to call the CLI SQLEndTran() function	32		

There are three core components of a CLI application: initialization, transaction processing, and termination. This chapter describes the key steps of a typical CLI application.

Initialization

Handles in CLI

A CLI handle is a variable that refers to a data object allocated and managed by DB2 CLI. Using handles relieves the application from having to allocate and manage global variables or data structures, such as the SQLDA.

There are four types of handles in CLI:

Environment handle

An environment handle refers to a data object that holds information about the global state of the application, such as attributes or valid connections. An environment handle must be allocated before a connection handle can be allocated.

Connection handle

A connection handle refers to a data object that holds information associated with a connection to a particular data source (database). Examples of such information include valid statement and descriptor handles on a connection, transaction status, and diagnostic information.

An application can be connected to several data sources at the same time, and can establish several distinct connections to the same data source. A separate connection handle must be allocated for each concurrent connection. A connection handle must be allocated before a statement or descriptor handle can be allocated.

Connection handles ensure that multithreaded applications which use one connection per thread are thread-safe, because separate data structures are allocated and maintained by DB2 CLI for each connection.

Note: There is a limit of 512 active connections per environment handle.

Statement handle

A statement handle refers to a data object that is used to track the execution of a single SQL statement. It provides access to statement information such as error messages, the associated cursor name, and status information for SQL statement processing. A statement handle must be allocated before an SQL statement can be issued.

When a statement handle is allocated, DB2 CLI automatically allocates four descriptors and assigns the handles for these descriptors to the `SQL_ATTR_APP_ROW_DESC`, `SQL_ATTR_APP_PARAM_DESC`, `SQL_ATTR_IMP_ROW_DESC`, and `SQL_ATTR_IMP_PARAM_DESC` statement attributes. Application descriptors can be explicitly allocated by allocating descriptor handles.

The number of statement handles available to a CLI application depends on the number of large packages the application has defined and is limited by overall system resources (usually stack size). By default, there are 3 small and 3 large packages. Each small package allows a maximum of 64 statement handles per connection, and each large package allows a maximum of 384 statement handles per connection. The number of available statement handles by default is therefore $(3 * 64) + (3 * 384) = 1344$.

To get more than the default 1344 statement handles, increase the number of large packages by setting the value of the CLI/ODBC configuration keyword `CLIPkg` to a value up to 30. `CLIPkg` indicates the number of large packages that will be generated. If you set `CLIPkg` to the maximum value of 30, then the maximum number of statement handles that is available becomes $(3 * 64) + (30 * 384) = 11\,712$.

An `HY014 SQLSTATE` may be returned on the call to `SQLPrepare()`, `SQLExecute()`, or `SQLExecuteDirect()` if this limit is exceeded.

It is recommended that you only allocate as many large packages as your application needs to run, as packages take up space in the database.

Descriptor handle

A descriptor handle refers to a data object that contains information about the columns in a result set and dynamic parameters in an SQL statement.

On operating systems that support multiple threads, applications can use the same environment, connection, statement, or descriptor handle on different threads. DB2 CLI provides thread safe access for all handles and function calls. The application itself might experience unpredictable behavior if the threads it creates do not co-ordinate their use of DB2 CLI resources.

Related concepts:

- “Descriptors in CLI applications” on page 151
- “Handle freeing in CLI applications” on page 38

Related tasks:

- “Allocating statement handles in CLI applications” on page 22
- “Freeing statement resources in CLI applications” on page 37
- “Initializing CLI applications” on page 18

Related reference:

- “SQLDA (SQL descriptor area)” in *SQL Reference, Volume 1*

- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLIPkg CLI/ODBC configuration keyword” on page 299
- “DB2 CLI bind files and package names” on page 203
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceRefreshInterval CLI/ODBC configuration keyword” on page 383

Initialization and termination in CLI overview

Figure 2 on page 18 shows the function call sequences for both the initialization and termination tasks. The transaction processing task in the middle of the diagram is shown in Transaction processing in CLI overview.

The initialization task consists of the allocation and initialization of environment and connection handles. An environment handle must be allocated before a connection handle can be created. Once a connection handle is created, the application can then establish a connection. When a connection exists, the application can proceed to the transaction processing task. An application then passes the appropriate handle when it calls other DB2 CLI functions.

The termination task consists of disconnecting from the data source and freeing those handles that were allocated during the initialization phase. The connection handle should be freed before freeing the environment handle.

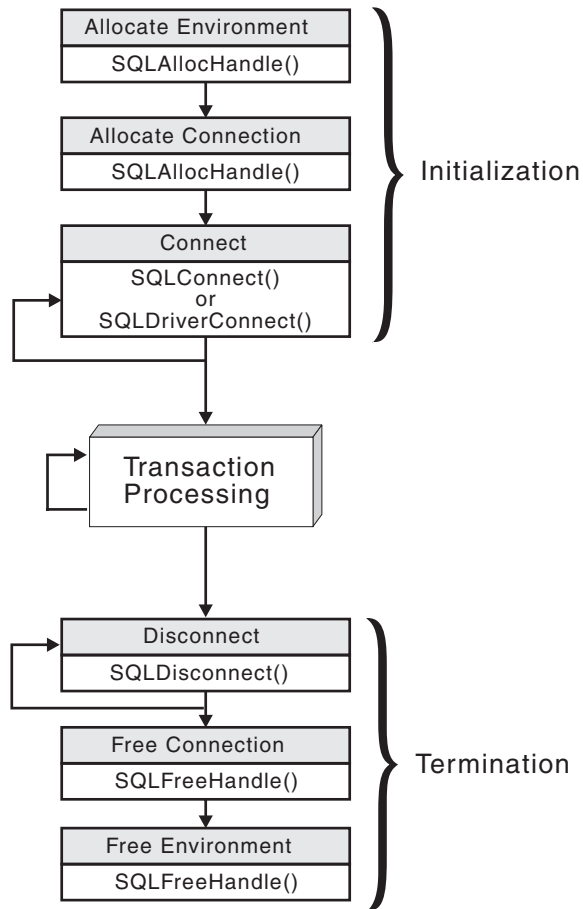


Figure 2. Conceptual view of initialization and termination tasks

Related concepts:

- “Handles in CLI” on page 15
- “Transaction processing in CLI overview” on page 20

Related reference:

- “SQLAllocHandle function (CLI) - Allocate handle” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLConnect function (CLI) - Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDisconnect function (CLI) - Disconnect from a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*

Initializing CLI applications

Initializing CLI applications is part of the larger task of programming with CLI. The task of initializing CLI applications involves allocating environment and connection handles and then connecting to the data source.

Procedure:

To initialize the application:

1. Allocate an environment handle by calling `SQLAllocHandle()` with a *HandleType* of `SQL_HANDLE_ENV` and an *InputHandle* of `SQL_NULL_HANDLE`. For example:

```
SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```

Use the allocated environment handle, returned in the **OutputHandlePtr* argument (henv in the example above), for all subsequent calls that require an environment handle.

2. Optional: Set environment attributes for your application by calling `SQLSetEnvAttr()` with the desired environment attribute for each attribute you want set.

Important: If you plan to run your application as an ODBC application, you must set the `SQL_ATTR_ODBC_VERSION` environment attribute using `SQLSetEnvAttr()`. Setting this attribute for applications that are strictly DB2 CLI applications is recommended but not required.

3. Allocate a connection handle by calling `SQLAllocHandle()` with a *HandleType* of `SQL_HANDLE_DBC` using the environment handle returned from Step 1 as the *InputHandle* argument. For example:

```
SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);
```

Use the allocated connection handle, returned in the **OutputHandlePtr* argument (hdbc in the example above), for all subsequent calls that require a connection handle.

4. Optional: Set connection attributes for your application by calling `SQLSetConnectAttr()` with the desired connection attribute for each attribute you want set.

5. Connect to a data source by calling one of following functions with the connection handle you allocated in Step 3 for each data source you want to connect to:

- `SQLConnect()`: basic database connection method. For example:
`SQLConnect (hdbc, server, SQL_NTS, user, SQL_NTS, password, SQL_NTS);`

where `SQL_NTS` is a special string length value that indicates the referenced string is null-terminated.

- `SQLDriverConnect()`: extended connect function that allows additional connect options and offers Graphical User Interface support. For example:

```
char * connStr = "DSN=SAMPLE;UID=;PWD=";
```

```
SQLDriverConnect (hdbc, (SQLHWND)NULL, connStr, SQL_NTS,  
                 NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
```

- `SQLBrowseConnect()`: least common connection method that iteratively returns the attributes and attribute values for connecting to a data source. For example:

```
char * connInStr = "DSN=SAMPLE;UID=;PWD=";  
char outStr[512];
```

```
SQLBrowseConnect (hdbc, connInStr, SQL_NTS, outStr,  
                 512, &strLen2Ptr);
```

Now that your application has been initialized, you can proceed to processing transactions.

Related concepts:

- “Handles in CLI” on page 15
- “Transaction processing in CLI overview” on page 20

Related reference:

- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLAllocHandle function (CLI) - Allocate handle” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBrowseConnect function (CLI) - Get required attributes to connect to data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLConnect function (CLI) - Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetEnvAttr function (CLI) - Set environment attribute” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “clihandl.c -- How to allocate and free handles”
- “dbcongui.c -- How to connect to a database with a graphical user interface (GUI)”
- “dbconn.c -- How to connect to and disconnect from a database”

Transaction processing

Transaction processing in CLI overview

Figure 3 on page 21 shows the typical order of function calls in the transaction processing task of a DB2 CLI application. Not all functions or possible paths are shown.

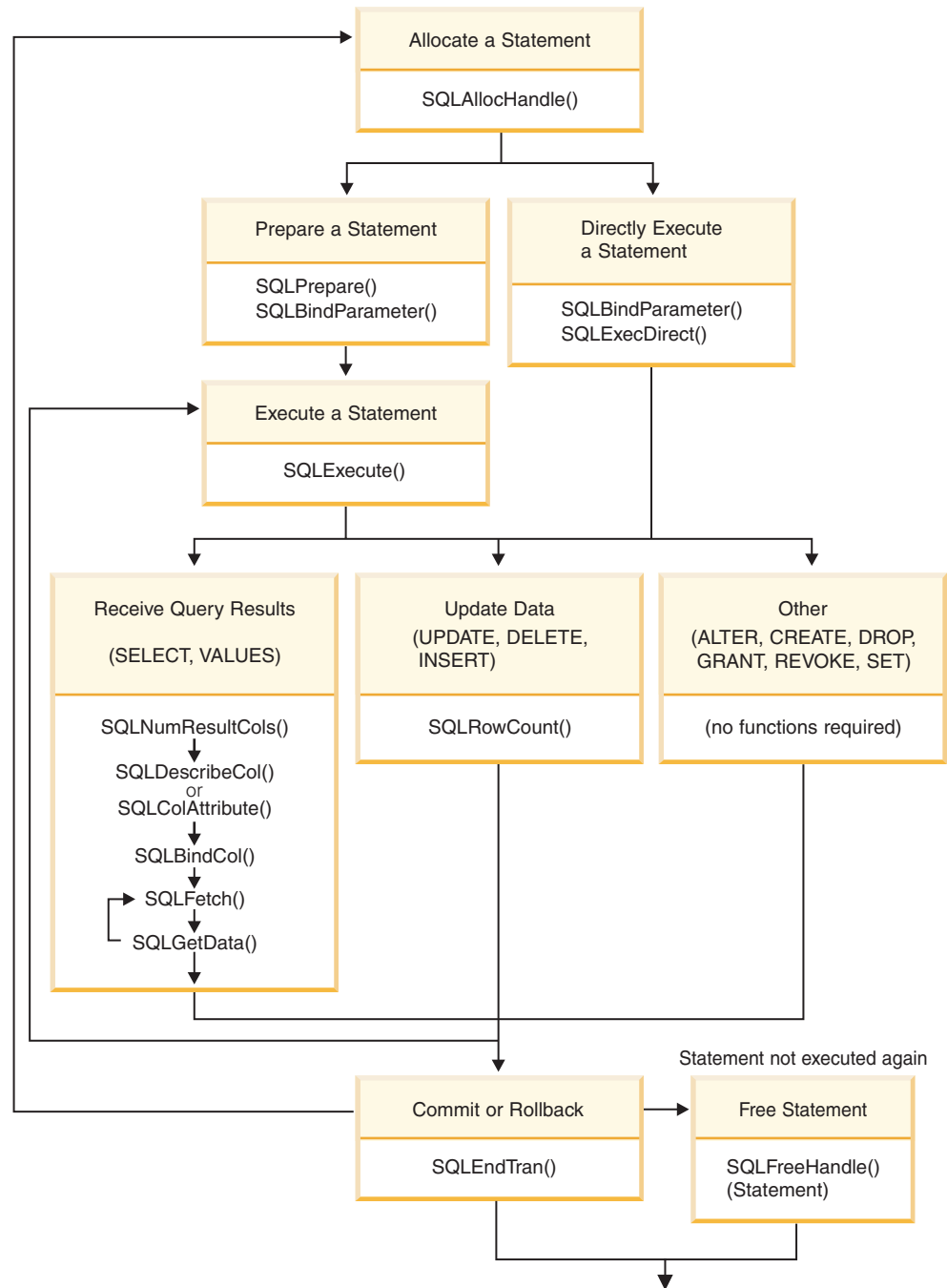


Figure 3. Transaction processing

The transaction processing task contains five steps:

- Allocating statement handle(s)
- Preparing and executing SQL statements
- Processing results
- Committing or Rolling Back
- (Optional) Freeing statement handle(s) if the statement is unlikely to be executed again.

Related concepts:

- “Commit modes in CLI applications” on page 30

Related tasks:

- “Allocating statement handles in CLI applications” on page 22
- “Freeing statement resources in CLI applications” on page 37
- “Preparing and executing SQL statements in CLI applications” on page 24
- “Retrieving query results in CLI applications” on page 33
- “Terminating a CLI application” on page 53
- “Updating and deleting data in CLI applications” on page 35

Allocating statement handles in CLI applications

To issue an SQL statement in a CLI application, you need to allocate a statement handle. A statement handle tracks the execution of a single SQL statement and is associated with a connection handle. Allocating statement handles is part of the larger task of processing transactions.

Prerequisites:

Before you begin allocating statement handles, you must allocate an environment handle and a connection handle. This is part of the task of initializing your CLI application.

Procedure:

To allocate a statement handle:

1. Call `SQLAllocHandle()` with a *HandleType* of `SQL_HANDLE_STMT`. For example:

```
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
```
2. Optional: To set attributes for this statement, call `SQLSetStmtAttr()` for each desired attribute option.

After allocating environment, connection, and statement handles, you can now prepare, issue, or execute SQL statements.

Related concepts:

- “Transaction processing in CLI overview” on page 20

Related tasks:

- “Initializing CLI applications” on page 18
- “Issuing SQL statements in CLI applications” on page 23
- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “SQLAllocHandle function (CLI) - Allocate handle” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “clihandl.c -- How to allocate and free handles”

Issuing SQL statements in CLI applications

SQL statements are passed to DB2 CLI functions as SQLCHAR string variables. The variable can consist of one or more SQL statements, with or without parameter markers, depending on the type of processing you want. This topic describes the various ways SQL statements can be issued in DB2 CLI applications.

Before you issue an SQL statement, ensure you have allocated a statement handle.

Procedure:

Perform either of the following steps to issue SQL statements:

- To issue a single SQL statement, either initialize an SQLCHAR variable with the SQL statement and pass this variable to the CLI function, or directly pass a string argument cast to an SQLCHAR * to the function. For example:

```
SQLCHAR * stmt = (SQLCHAR *) "SELECT deptname, location FROM org";
/* ... */
SQLExecDirect (hstmt, stmt, SQL_NTS);
```

or

```
SQLExecDirect (hstmt, (SQLCHAR *) "SELECT deptname, location FROM org",
               SQL_NTS);
```

- To issue multiple SQL statements on the same statement handle, either initialize an array of SQLCHAR elements, where each element represents an individual SQL statement, or initialize a single SQLCHAR variable that contains the multiple statements delimited by a ";" character. For example:

```
SQLCHAR * multiple_stmts[] = {
    (SQLCHAR *) "SELECT deptname, location FROM org",
    (SQLCHAR *) "SELECT id, name FROM staff WHERE years > 5",
    (SQLCHAR *) "INSERT INTO org VALUES (99, 'Hudson', 20, 'Western', 'Seattle')";
};
```

or

```
SQLCHAR * multiple_stmts =
"SELECT deptname, location FROM org;
SELECT id, name FROM staff WHERE years > 5;
INSERT INTO org VALUES (99, 'Hudson', 20, 'Western', 'Seattle');";
```

Note: When a list of SQL statements is specified, only one statement is executed at a time, starting with the first statement in the list. Each subsequent statement is executed in the order it appears. (To execute subsequent statements, you must call `SQLMoreResults()`.)

- To issue SQL statements with parameter markers, see [Binding Parameter Markers](#).
- To capture and convert SQL statements dynamically executed with DB2 CLI (dynamic SQL) to static SQL, see [Creating Static SQL](#).

Related concepts:

- [“Parameter marker binding in CLI applications”](#) on page 26

Related tasks:

- [“Binding parameter markers in CLI applications”](#) on page 29
- [“Creating static SQL with CLI/ODBC/JDBC Static Profiling”](#) on page 185
- [“Allocating statement handles in CLI applications”](#) on page 22

Related reference:

- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLMoreResults function (CLI) - Determine if there are more result sets” in *Call Level Interface Guide and Reference, Volume 2*
- “C data types for CLI applications” on page 43

Related samples:

- “dbuse.c -- How to use a database”

Preparing and executing SQL statements in CLI applications

Once you have allocated a statement handle, you can then perform operations using SQL statements or XQuery expressions. An SQL statement or XQuery expression must be prepared before it can be executed, and DB2 CLI offers two ways of preparing and executing:

- perform the prepare and execute operations in separate steps
- combine the prepare and execute operations into one step

Prerequisites:

Before preparing and executing your SQL statement or XQuery expression, ensure you have allocated a statement handle for it.

Procedure:

To prepare and execute an SQL statement or XQuery expression in separate steps:

1. Prepare the SQL statement or XQuery expression by calling `SQLPrepare()` and passing the statement or expression as the *StatementText* argument.

Note: XQuery expressions must be prefixed with the case-insensitive “XQUERY” keyword, unless the statement attribute `SQL_ATTR_XQUERY_STATEMENT` has been set to `SQL_TRUE` for this statement handle.

2. Call `SQLBindParameter()` to bind any parameter markers you may have in the SQL statement.

Note: For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the `XMLQUERY` function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in `XMLQUERY` for execution.

3. Execute the prepared statement by calling `SQLExecute()`.

Use this method when:

- The same SQL statement or XQuery expression will be executed repeatedly (usually with different parameter values). This avoids having to prepare the same statement or expression more than once. The subsequent executions make use of the access plans already generated by the prepare, thus increasing driver efficiency and delivering better application performance.
- The application requires information about the parameters or columns in the result set prior to statement execution.

To prepare and execute an SQL statement or XQuery expression in one step:

1. Call `SQLBindParameter()` to bind any parameter markers you may have in the SQL statement.

Note: For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the `XMLQUERY` function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in `XMLQUERY` for execution.

2. Prepare and execute the statement or expression by calling `SQLExecDirect()` with the SQL statement or XQuery expression as the *StatementText* argument.

Note: XQuery expressions must be prefixed with the case-insensitive "XQUERY" keyword, unless the statement attribute `SQL_ATTR_XQUERY_STATEMENT` has been set to `SQL_TRUE` for this statement handle.

3. Optional: If a list of SQL statements are to be executed, call `SQLMoreResults()` to advance to the next SQL statement.

Use this method of preparing and executing in one step when:

- The statement or expression will be executed only once. This avoids having to call two functions to execute the statement or expression.
- The application does not require information about the columns in the result set before the statement is executed.

Related concepts:

- "Deferred prepare in CLI applications" on page 26
- "Transaction processing in CLI overview" on page 20

Related tasks:

- "Allocating statement handles in CLI applications" on page 22

Related reference:

- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLExecDirect function (CLI) - Execute a statement directly" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLExecute function (CLI) - Execute a statement" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLMoreResults function (CLI) - Determine if there are more result sets" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLPrepare function (CLI) - Prepare a statement" in *Call Level Interface Guide and Reference, Volume 2*
- "Statement attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*
- "XMLQUERY scalar function" in *SQL Reference, Volume 1*

Related samples:

- "dbuse.c -- How to use a database"

Deferred prepare in CLI applications

Deferred prepare is the name of the CLI feature that seeks to minimise communication with the server by sending both the prepare and execute requests for SQL statements in the same network flow. The default value for this property can be overridden using the CLI/ODBC configuration keyword `DeferredPrepare`. This property can be set on a per-statement handle basis by calling `SQLSetStmtAttr()` to change the `SQL_ATTR_DEFERRED_PREPARE` statement attribute.

When deferred prepare is on, the prepare request is not sent to the server until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance. Because of this behavior, any errors that would typically be generated by `SQLPrepare()` will appear at execute time, and `SQLPrepare()` will always return `SQL_SUCCESS`. Deferred prepare is of greatest benefit when the application generates queries where the answer set is very small, and the overhead of separate requests and replies is not spread across multiple blocks of query data.

Note: Even if deferred prepare is enabled, operations that require a statement to be prepared prior to the operation's execution will force the prepare request to be sent to the server before the execute. Describe operations resulting from calls to `SQLDescribeParam()` or `SQLDescribeCol()` are examples of when deferred prepare will be overridden, because describe information is only available after the statement has been prepared.

Related tasks:

- "Preparing and executing SQL statements in CLI applications" on page 24

Related reference:

- "CLI function return codes" on page 50
- "DeferredPrepare CLI/ODBC configuration keyword" on page 316
- "SQLDescribeCol function (CLI) - Return a set of attributes for a column" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLDescribeParam function (CLI) - Return description of a parameter marker" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLPrepare function (CLI) - Prepare a statement" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" in *Call Level Interface Guide and Reference, Volume 2*
- "Statement attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*

Parameter marker binding in CLI applications

Parameter markers are represented by the '?' character and indicate the position in the SQL statement where the contents of application variables are to be substituted when the statement is executed. (A parameter marker is used where a host variable would be used in static embedded SQL.) This value can be obtained from:

- An application variable.
`SQLBindParameter()` is used to bind the application storage area to the parameter marker.
- A LOB value from the database server (by specifying a LOB locator).

SQLBindParameter() is used to bind a LOB locator to the parameter marker. The LOB value itself is supplied by the database server, so only the LOB locator is transferred between the database server and the application.

- A file within the application's environment containing a LOB value.

SQLBindFileToParam() is used to bind a file to a LOB parameter marker. When SQLExecDirect() is executed, DB2 CLI will transfer the contents of the file directly to the database server.

Parameter markers are referenced sequentially, from left to right, starting at 1. SQLNumParams() can be used to determine the number of parameters in a statement.

The application must bind an application variable to each parameter marker in the SQL statement before it executes that statement. Binding is carried out by calling the SQLBindParameter() function with a number of arguments to indicate:

- the ordinal position of the parameter,
- the SQL type of the parameter,
- the type of parameter (input, output, or inout),
- the C data type of the variable,
- a pointer to the application variable,
- the length of the variable.

The bound application variable and its associated length are called *deferred* input arguments because only the pointers are passed when the parameter is bound; no data is read from the variable until the statement is executed. Deferred arguments allow the application to modify the contents of the bound parameter variables, and re-execute the statement with the new values.

Information about each parameter remains in effect until:

- it is overridden by the application
- the application unbinds the parameter by calling SQLFreeStmt() with the SQL_RESET_PARAMS *Option*
- the application drops the statement handle by calling SQLFreeHandle() with a *HandleType* of SQL_HANDLE_STMT or SQLFreeStmt() with the SQL_DROP *Option*.

Information for each parameter remains in effect until overridden, or until the application unbinds the parameter or drops the statement handle. If the application executes the SQL statement repeatedly without changing the parameter binding, then DB2 CLI uses the same pointers to locate the data on each execution. The application can also change the parameter binding to a different set of deferred variables by calling SQLBindParameter() again for one or more parameters and specifying different application variables. The application must not deallocate or discard variables used for deferred input fields between the time it binds the fields to parameter markers and the time DB2 CLI accesses them at execution time. Doing so can result in DB2 CLI reading garbage data, or accessing invalid memory resulting in an application trap.

It is possible to bind the parameter to a variable of a different type from that required by the SQL statement. The application must indicate the C data type of the source, and the SQL type of the parameter marker, and DB2 CLI will convert the contents of the variable to match the SQL data type specified. For example, the SQL statement may require an integer value, but your application has a string

representation of an integer. The string can be bound to the parameter, and DB2 CLI will convert the string to the corresponding integer value when you execute the statement.

By default, DB2 CLI does not verify the type of the parameter marker. If the application indicates an incorrect type for the parameter marker, it could cause:

- an extra conversion by the DBMS
- an error at the DBMS which forces DB2 CLI to describe the statement being executed and re-execute it, resulting in extra network traffic
- an error returned to the application if the statement cannot be described, or the statement cannot be re-executed successfully.

Information about the parameter markers can be accessed using descriptors. If you enable automatic population of the implementation parameter descriptor (IPD) then information about the parameter markers will be collected. The statement attribute `SQL_ATTR_ENABLE_AUTO_IPD` must be set to `SQL_TRUE` for this to work.

If the parameter marker is part of a predicate on a query and is associated with a User Defined Type, then the parameter marker must be cast to the built-in type in the predicate portion of the statement; otherwise, an error will occur.

After the SQL statement has been executed, and the results processed, the application may wish to reuse the statement handle to execute a different SQL statement. If the parameter marker specifications are different (number of parameters, length or type) then `SQLFreeStmt()` should be called with `SQL_RESET_PARAMS` to reset or clear the parameter bindings.

Related concepts:

- “Data types and data conversion in CLI applications” on page 39
- “Descriptors in CLI applications” on page 151
- “Handles in CLI” on page 15
- “Large object usage in CLI applications” on page 96
- “User-defined type (UDT) usage in CLI applications” on page 148

Related reference:

- “SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeStmt function (CLI) - Free (or reset) a statement handle” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLNumParams function (CLI) - Get number of parameters in a SQL statement” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbuse.c -- How to use a database”
- “dtlob.c -- How to read and write LOB data”
- “spclient.c -- Call various stored procedures”
- “tbmod.c -- How to modify table data”

Binding parameter markers in CLI applications

This topic describes how to bind parameter markers to application variables before executing SQL statements. Parameter markers in SQL statements can be bound to single values or to arrays of values. Binding each parameter marker individually requires a network flow to the server for each set of values. Using arrays, however, allows several sets of parameter values to be bound and sent at once to the server.

Prerequisites:

Before you bind parameter markers, ensure you have initialized your application.

Procedure:

To bind parameter markers, perform either of the following steps:

- To bind parameter markers one at a time to application variables, call `SQLBindParameter()` for each application variable you want to bind. Ensure you specify the correct parameter type: `SQL_PARAM_INPUT`, `SQL_PARAM_OUTPUT`, or `SQL_PARAM_INPUT_OUTPUT`. The following example shows how two parameter markers are bound with two application variables:

```
SQLCHAR *stmt =
    (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? AND division = ? ";
SQLSMALLINT parameter1 = 0;
char parameter2[20];
```

```
/* bind parameter1 to the statement */
cliRC = SQLBindParameter(hstmt,
    1,
    SQL_PARAM_INPUT,
    SQL_C_SHORT,
    SQL_SMALLINT,
    0,
    0,
    &parameter1,
    0,
    NULL);
```

```
/* bind parameter2 to the statement */
cliRC = SQLBindParameter(hstmt,
    2,
    SQL_PARAM_INPUT,
    SQL_C_CHAR,
    SQL_VARCHAR,
    20,
    0,
    parameter2,
    20,
    NULL);
```

- To bind at once many values to parameter markers, perform either of the following tasks which use arrays of values:
 - binding parameter markers with column-wise array input
 - binding parameter markers with row-wise array input

Related concepts:

- “Parameter marker binding in CLI applications” on page 26

Related tasks:

- “Binding parameter markers in CLI applications with column-wise array input” on page 81
- “Binding parameter markers in CLI applications with row-wise array input” on page 82
- “Initializing CLI applications” on page 18

Related reference:

- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbuse.c -- How to use a database”
- “tbmod.c -- How to modify table data”

Commit modes in CLI applications

A *transaction* is a recoverable unit of work, or a group of SQL statements that can be treated as one atomic operation. This means that all the operations within the group are guaranteed to be completed (committed) or undone (rolled back), as if they were a single operation. When the transaction spans multiple connections, it is referred to as a distributed unit of work (DUOW).

Transactions are started implicitly with the first access to the database using `SQLPrepare()`, `SQLExecDirect()`, `SQLGetTypeInfo()`, or any function that returns a result set, such as catalog functions. At this point a transaction has begun, even if the call failed.

DB2 CLI supports two commit modes:

auto-commit

In auto-commit mode, every SQL statement is a complete transaction, which is automatically committed. For a non-query statement, the commit is issued at the end of statement execution. For a query statement, the commit is issued after the cursor has been closed. The default commit mode is auto-commit (except when participating in a coordinated transaction).

manual-commit

In manual-commit mode, the transaction ends when you use `SQLEndTran()` to either rollback or commit the transaction. This means that any statements executed (on the same connection) between the start of a transaction and the call to `SQLEndTran()` are treated as a single transaction. If DB2 CLI is in manual-commit mode, a new transaction is implicitly started when an SQL statement that can be contained within a transaction is executed against the current data source.

An application can switch between manual-commit and auto-commit modes by calling `SQLSetConnectAttr()`. Auto-commit can be useful for query-only applications, because the commits can be chained to the SQL execution request sent to the server. Another benefit of auto-commit is improved concurrency since locks are removed as soon as possible. Applications that need to perform updates to the database should turn off auto-commit as soon as the database connection has been established and should not wait until the disconnect before committing or rolling back the transaction.

The following are examples of how to set auto-commit on and off:

- Setting auto-commit on:

```
/* ... */

/* set AUTOCOMMIT on */
sqlrc = SQLSetConnectAttr( hdbc,
                           SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)SQL_AUTOCOMMIT_ON, SQL_NTS ) ;

/* continue with SQL statement execution */
```

- Setting auto-commit off:

```
/* ... */

/* set AUTOCOMMIT OFF */
sqlrc = SQLSetConnectAttr( hdbc,
                           SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_NTS ) ;

/* ... */

/* execute the statement */
/* ... */
sqlrc = SQLExecDirect( hstmt, stmt, SQL_NTS ) ;

/* ... */

sqlrc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK ) ;
DBC_HANDLE_CHECK( hdbc, sqlrc);

/* ... */
```

When multiple connections exist to the same or different databases, each connection has its own transaction. Special care must be taken to call `SQLEndTran()` with the correct connection handle to ensure that only the intended connection and related transaction is affected. It is also possible to rollback or commit all the connections by specifying a valid environment handle, and a NULL connection handle on the `SQLEndTran()` call. Unlike distributed unit of work connections, there is no coordination between the transactions on each connection in this case.

Related concepts:

- “Catalog functions for querying system catalog information in CLI applications” on page 167
- “Cursors in CLI applications” on page 65
- “DB2 as transaction manager in CLI applications” on page 138
- “Multisite updates (two phase commit) in CLI applications” on page 137

Related tasks:

- “Calling stored procedures from CLI applications” on page 113
- “Updating and deleting data in CLI applications” on page 35

Related reference:

- “SQLEndTran function (CLI) - End transactions of a connection or an Environment” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetTypeInfo function (CLI) - Get data type information” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*

When to call the CLI `SQLEndTran()` function

In auto-commit mode, a commit is issued implicitly at the end of each statement execution or when a cursor is closed.

In manual-commit mode, `SQLEndTran()` must be called before calling `SQLDisconnect()`. If a Distributed Unit of Work is involved, additional rules may apply.

Consider the following when deciding where in the application to end a transaction:

- Each connection cannot have more than one current transaction at any given time, so keep dependent statements within the same unit of work. Note that statements must always be kept on the same connection under which they were allocated.
- Various resources may be held while the current transaction on a connection is running. Ending the transaction will release the resources for use by other applications.
- Once a transaction has successfully been committed or rolled back, it is fully recoverable from the system logs. Open transactions are not recoverable.

Effects of calling `SQLEndTran()`:

When a transaction ends:

- All locks on DBMS objects are released, except those that are associated with a held cursor.
- Prepared statements are preserved from one transaction to the next. Once a statement has been prepared on a specific statement handle, it does not need to be prepared again even after a commit or rollback, provided the statement continues to be associated with the same statement handle.
- Cursor names, bound parameters, and column bindings are maintained from one transaction to the next.
- By default, cursors are preserved after a commit (but not a rollback). All cursors are by default defined with the `WITH HOLD` clause, except when the CLI application is running in a Distributed Unit of Work environment.

Related concepts:

- “Column binding in CLI applications” on page 85
- “Commit modes in CLI applications” on page 30
- “Cursors in CLI applications” on page 65
- “Handles in CLI” on page 15
- “Multisite updates (two phase commit) in CLI applications” on page 137
- “Parameter marker binding in CLI applications” on page 26

Related reference:

- “SQLDisconnect function (CLI) - Disconnect from a data source” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLEndTran function (CLI) - End transactions of a connection or an Environment” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbmcon.c -- How to use multiple databases”
- “dbuse.c -- How to use a database”

Retrieving query results in CLI applications

Retrieving query results is part of the larger task of processing transactions in CLI applications. Retrieving query results involves binding application variables to columns of a result set and then fetching the rows of data into the application variables. A typical query is the SELECT statement.

Prerequisites:

Before you retrieve results, ensure you have initialized your application and prepared and executed the necessary SQL statements.

Procedure:

To retrieve each row of the result set:

1. Optional: Determine the structure of the result set, number of columns, and column types and lengths by calling `SQLNumResultCols()` and `SQLDescribeCol()`.

Note: Performing this step can reduce performance if done before the query has been executed, because it forces CLI to describe the query’s columns. Information about the result set’s columns is available after successful execution, and describing the result set does not incur any additional overhead if the describe is performed after successful execution.

2. Bind an application variable to each column of the result set, by calling `SQLBindCol()`, ensuring that the variable type matches the column type. For example:

```

struct
{
    SQLINTEGER ind;
    SQLSMALLINT val;
}
deptnumb; /* variable to be bound to the DEPTNUMB column */

struct
{
    SQLINTEGER ind;
    SQLCHAR val[15];
}
location; /* variable to be bound to the LOCATION column */

/* ... */

/* bind column 1 to variable */
cliRC = SQLBindCol(hstmt, 1, SQL_C_SHORT, &deptnumb.val, 0,
                  &deptnumb.ind);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

```

```

/* bind column 2 to variable */
cliRC = SQLBindCol(hstmt, 2, SQL_C_CHAR, location.val, 15,
                  &location.ind);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

```

The application can use the information obtained in step 1 to determine an appropriate C data type for the application variable and to allocate the maximum storage the column value could occupy. The columns are bound to deferred output arguments, which means the data is written to these storage locations when it is fetched.

Important: Do not de-allocate or discard variables used for deferred output arguments between the time the application binds them to columns of the result set and the time DB2 CLI writes to these arguments.

3. Repeatedly fetch the row of data from the result set by calling `SQLFetch()` until `SQL_NO_DATA_FOUND` is returned. For example:

```

/* fetch each row and display */
cliRC = SQLFetch(hstmt);

if (cliRC == SQL_NO_DATA_FOUND)
{
    printf("\n Data not found.\n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
    printf("    %-8d %-14.14s \n", deptnumb.val, location.val);

    /* fetch next row */
    cliRC = SQLFetch(hstmt);
}

```

`SQLFetchScroll()` can also be used to fetch multiple rows of the result set into an array.

If data conversion was required for the data types specified on the call to `SQLBindCol()`, the conversion will occur when `SQLFetch()` is called.

4. Optional: Retrieve columns that were not previously bound by calling `SQLGetData()` after each successful fetch. You can retrieve all unbound columns this way. For example:

```

/* fetch each row and display */
cliRC = SQLFetch(hstmt);

if (cliRC == SQL_NO_DATA_FOUND)
{
    printf("\n Data not found.\n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
    /* use SQLGetData() to get the results */
    /* get data from column 1 */
    cliRC = SQLGetData(hstmt,
                      1,
                      SQL_C_SHORT,
                      &deptnumb.val,
                      0,
                      &deptnumb.ind);
    STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

    /* get data from column 2 */
    cliRC = SQLGetData(hstmt,
                      2,
                      SQL_C_CHAR,
                      location.val,
                      15,

```

```

                                &location.ind);

/* display the data */
printf("    %-8d %-14.14s \n", deptnumb.val, location.val);

/* fetch the next row */
cliRC = SQLFetch(hstmt);
}

```

Note: Applications perform better if columns are bound, rather than having them retrieved as unbound columns using `SQLGetData()`. However, an application may be constrained in the amount of long data it can retrieve and handle at one time. If this is a concern, then `SQLGetData()` may be the better choice.

Related concepts:

- “Data types and data conversion in CLI applications” on page 39

Related tasks:

- “Initializing CLI applications” on page 18
- “Preparing and executing SQL statements in CLI applications” on page 24
- “Retrieving array data in CLI applications using column-wise binding” on page 89
- “Retrieving array data in CLI applications using row-wise binding” on page 90

Related reference:

- “C data types for CLI applications” on page 43
- “CLI function return codes” on page 50
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLNumResultCols function (CLI) - Get number of result columns” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “tbread.c -- How to read data from tables”

Updating and deleting data in CLI applications

Updating and deleting data is part of the larger task of processing transactions in CLI. There are two types of update and delete operations available in CLI programming: simple and positioned. A simple update or delete operation only requires that you issue and execute the `UPDATE` or `DELETE` SQL statements as you would any other SQL statement. You could, in this case, use `SQLRowCount()` to obtain the number of rows affected by the SQL statement.

Positioned updates and deletes involve modifying the data of a result set. A positioned update is the update of a column of a result set, and a positioned delete is when a row of a result set is deleted. Positioned update and delete operations require cursors to be used. This document describes how to perform positioned update and delete operations by first getting the name of the cursor associated with the result set, and then issuing and executing the UPDATE or DELETE on a second statement handle using the retrieved cursor name.

Prerequisites:

Before you perform a positioned update or delete operation, ensure that you have initialized your CLI application.

Procedure:

To perform a positioned update or delete operation:

1. Generate the result set that the update or delete will be performed on by issuing and executing the SELECT SQL statement.
2. Call `SQLGetCursorName()` to get the name of the cursor, using the same statement handle as the handle that executed the SELECT statement. This cursor name will be needed in the UPDATE or DELETE statement.

When a statement handle is allocated, a cursor name is automatically generated. You can define your own cursor name using `SQLSetCursorName()`, but it is recommended that you use the name that is generated by default because all error messages will reference the generated name, not the name defined using `SQLSetCursorName()`.

3. Allocate a second statement handle that will be used to execute the positioned update or delete.

To update a row that has been fetched, the application uses two statement handles, one for the fetch and one for the update. You cannot reuse the fetch statement handle to execute the positioned update or delete, because it is still in use when the positioned update or delete is executing.

4. Fetch data from the result set by calling `SQLFetch()` or `SQLFetchScroll()`.
5. Issue the UPDATE or DELETE SQL statement with the WHERE CURRENT of clause and specify the cursor name obtained in step 2. For example:

```
sprintf((char *)stmtPositionedUpdate,
        "UPDATE org SET location = 'Toronto' WHERE CURRENT of %s",
        cursorName);
```

6. Position the cursor on the row of the data fetched and execute the positioned update or delete statement.

Related tasks:

- “Initializing CLI applications” on page 18
- “Issuing SQL statements in CLI applications” on page 23
- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “DELETE statement” in *SQL Reference, Volume 2*
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLGetCursorName function (CLI) - Get cursor name” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLRowCount function (CLI) - Get row count” in *Call Level Interface Guide and Reference, Volume 2*
- “UPDATE statement” in *SQL Reference, Volume 2*

Related samples:

- “spserver.c -- Definition of various types of stored procedures”
- “tbmod.c -- How to modify table data”

Freeing statement resources in CLI applications

After a transaction has completed, end the processing for each statement handle by freeing the resources associated with it. There are four main tasks that are involved with freeing resources for a statement handle:

- close the open cursor
- unbind the column bindings
- unbind the parameter bindings
- free the statement handle

There are two ways you can free statement resources: using `SQLFreeHandle()` or `SQLFreeStmt()`.

Prerequisites:

Before you can free statement resources, you must have initialized your CLI application and allocated a statement handle.

Procedure:

To free statement resources with `SQLFreeHandle()`, call `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_STMT` and the handle you want to free. This will close any open cursor associated with this statement handle, unbind column and parameter bindings, and free the statement handle. This invalidates the statement handle. You do not need to explicitly carry out each of the four tasks listed above.

To free statement resources with `SQLFreeStmt()`, you need to call `SQLFreeStmt()` for each task (depending on how the application was implemented, all of these tasks may not be necessary):

- To close the open cursor, call `SQLCloseCursor()`, or call `SQLFreeStmt()` with the `SQL_CLOSE` *Option* and statement handle as arguments. This closes the cursor and discards any pending results.
- To unbind column bindings, call `SQLFreeStmt()` with an *Option* of `SQL_UNBIND` and the statement handle. This unbinds all columns for this statement handle except the bookmark column.
- To unbind parameter bindings, call `SQLFreeStmt()` with an *Option* of `SQL_RESET_PARAMS` and the statement handle. This releases all parameter bindings for this statement handle.
- To free the statement handle, call `SQLFreeStmt()` with an *Option* of `SQL_DROP` and the statement handle to be freed. This invalidates this statement handle.

Note: Although this option is still supported, we recommend that you use `SQLFreeHandle()` in your DB2 CLI applications so that they conform to the latest standards.

Related concepts:

- “Handle freeing in CLI applications” on page 38
- “Handles in CLI” on page 15

Related tasks:

- “Allocating statement handles in CLI applications” on page 22
- “Initializing CLI applications” on page 18

Related reference:

- “SQLCloseCursor function (CLI) - Close cursor and discard pending results” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeStmt function (CLI) - Free (or reset) a statement handle” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “utilcli.c -- Utility functions used by DB2 CLI samples”

Handle freeing in CLI applications

Environment handle:

Prior to calling `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_ENV`, an application must call `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_DBC` for all connections allocated under the environment. Otherwise, the call to `SQLFreeHandle()` returns `SQL_ERROR` and the environment remains valid, as well as any connection associated with that environment.

Connection handle:

If a connection is open on the handle, an application must call `SQLDisconnect()` for the connection prior to calling `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_DBC`. Otherwise, the call to `SQLFreeHandle()` returns `SQL_ERROR` and the connection remains valid.

Statement handle:

A call to `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_STMT` frees all resources that were allocated by a call to `SQLAllocHandle()` with a *HandleType* of `SQL_HANDLE_STMT`. When an application calls `SQLFreeHandle()` to free a statement that has pending results, the pending results are discarded. When an application frees a statement handle, DB2 CLI frees all the automatically generated descriptors associated with that handle.

Note that `SQLDisconnect()` automatically drops any statements and descriptors open on the connection.

Descriptor Handle:

A call to `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_DESC` frees the descriptor handle in *Handle*. The call to `SQLFreeHandle()` does not release any memory allocated by the application that may be referenced by the deferred fields (`SQL_DESC_DATA_PTR`, `SQL_DESC_INDICATOR_PTR`, and `SQL_DESC_OCTET_LENGTH_PTR`) of any descriptor record of *Handle*. When an explicitly allocated descriptor handle is freed, all statements that the freed handle had been associated with revert to their automatically allocated descriptor handle.

Note that `SQLDisconnect()` automatically drops any statements and descriptors open on the connection. When an application frees a statement handle, DB2 CLI frees all the automatically generated descriptors associated with that handle.

Related concepts:

- “Descriptors in CLI applications” on page 151

Related tasks:

- “Freeing statement resources in CLI applications” on page 37
- “Terminating a CLI application” on page 53

Related reference:

- “CLI function return codes” on page 50
- “Descriptor header and record field initialization values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDisconnect function (CLI) - Disconnect from a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*

Data types and data conversion in CLI applications

When writing a DB2 CLI application it is necessary to work with both SQL data types and C data types. This is unavoidable because the DBMS uses SQL data types, while the application uses C data types. The application, therefore, must match C data types to SQL data types when calling DB2 CLI functions to transfer data between the DBMS and the application.

To facilitate this, DB2 CLI provides symbolic names for the various data types, and manages the transfer of data between the DBMS and the application. It also performs data conversion (from a C character string to an SQL INTEGER type, for example) if required. DB2 CLI needs to know both the source and target data type. This requires the application to identify both data types using symbolic names.

Data type conversion can occur under one of two conditions:

- The application specified a C type that is not the default C type for the SQL type.
- The application specified an SQL type that does not match the base column SQL type at the server, and there was no describe information available to the DB2 CLI driver.

Example of how to use data types:

Because the data source contains SQL data types and the CLI application works with C data types, the data to be retrieved needs to be handled with the correct

data types. The following example shows how SQL and C data types are used by an application to retrieve data from the source into application variables. The example is taken from the `tut_read.c` sample program and examines how data is retrieved from the DEPTNUMB column of the ORG table in the sample database.

- The DEPTNUMB column of the ORG table is declared as the SQL data type SMALLINT.
- The application variable which will hold the retrieved data is declared using C types. Since the DEPTNUMB column is of SQL type SMALLINT, the application variable needs to be declared using the C type SQLSMALLINT, which is equivalent to the SQL type SMALLINT.

```
struct
{
    SQLINTEGER ind;
    SQLSMALLINT val;
} deptnumb;      /* variable to be bound to the DEPTNUMB column */
```

SQLSMALLINT represents the base C type of short int.

- The application binds the application variable to the symbolic C data type of SQL_C_SHORT:

```
sqlrc = SQLBindCol(hstmt, 1, SQL_C_SHORT, &deptnumb.val, 0,
                  &deptnumb.ind);
```

The data types are now consistent, because the result data type SQL_C_SHORT represents the C type SQLSMALLINT.

Data conversion:

DB2 CLI manages the transfer and any required conversion of data between the application and the DBMS. Before the data transfer actually takes place, either the source, the target or both data types are indicated when calling `SQLBindParameter()`, `SQLBindCol()` or `SQLGetData()`. These functions use the symbolic type names to identify the data types involved.

For example, to bind a parameter marker that corresponds to an SQL data type of DECIMAL(5,3), to an application's C buffer type of double, the appropriate `SQLBindParameter()` call would look like:

```
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
                  SQL_DECIMAL, 5, 3, double_ptr, 0, NULL);
```

The functions mentioned in the previous paragraph can be used to convert data from the default to other data types, but not all data conversions are supported or make sense.

The rules that specify limits on precision and scale, as well as truncation and rounding rules for type conversions apply in DB2 CLI, with the following exception: truncation of values to the right of the decimal point for numeric values may return a truncation warning, whereas truncation to the left of the decimal point returns an error. In cases of error, the application should call `SQLGetDiagRec()` to obtain the SQLSTATE and additional information on the failure. When moving and converting floating point data values between the application and DB2 CLI, no correspondence is guaranteed to be exact as the values may change in precision and scale.

Related concepts:

- "SQLSTATES for DB2 CLI" on page 51

Related reference:

- “C data types for CLI applications” on page 43
- “Data conversions supported in CLI” on page 397
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtinfo.c -- How get information about data types”

SQL symbolic and default data types for CLI applications

The table below lists each of the SQL data types used by CLI applications, with its corresponding symbolic name, and the default C symbolic name.

SQL data type

This column contains the SQL data types as they would appear in an SQL CREATE statement. The SQL data types are dependent on the DBMS.

Symbolic SQL data type

This column contains SQL symbolic names that are defined (in `sqlcli.h`) as an integer value. These values are used by various functions to identify the SQL data types listed in the first column.

Default C symbolic data type

This column contains C symbolic names, also defined as integer values. These values are used in various function arguments to identify the C data type. The symbolic names are used by various functions, such as `SQLBindParameter()`, `SQLGetData()`, and `SQLBindCol()` to indicate the C data types of the application variables. Instead of explicitly identifying C data types when calling these functions, `SQL_C_DEFAULT` can be specified instead, and DB2 CLI will assume a default C data type based on the SQL data type of the parameter or column as shown by this table. For example, the default C data type of `SQL_DECIMAL` is `SQL_C_CHAR`.

It is recommended that applications do not use `SQL_C_DEFAULT` to define C data types because it is less efficient for the CLI driver. Explicitly indicating the C data type in the application is preferred since it yields better performance than using `SQL_C_DEFAULT`.

Table 3. SQL symbolic and default data types

SQL data type	Symbolic SQL data type	Default symbolic C data type
BIGINT	SQL_BIGINT	SQL_C_SBIGINT
BINARY	SQL_BINARY	SQL_C_BINARY
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR ^a	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
CHAR	SQL_CHAR	SQL_C_CHAR

Table 3. SQL symbolic and default data types (continued)

SQL data type	Symbolic SQL data type	Default symbolic C data type
CHAR	SQL_TINYINT	SQL_C_TINYINT
CHAR FOR BIT DATA ^b	SQL_BINARY	SQL_C_BINARY
CHAR FOR BIT DATA	SQL_BIT	SQL_C_BINARY
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR ^a	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
DATALINK	SQL_DATALINK	SQL_C_CHAR
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR
DBCLOB LOCATOR ^a	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DECFLOAT(16)	SQL_DECFLOAT	SQL_C_CHAR
DECFLOAT(34)	SQL_DECFLOAT	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR ^b	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA ^b	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC ^b	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
LONG VARGRAPHIC ^b	SQL_WLONGVARCHAR	SQL_C_DBCHAR
NUMERIC ^c	SQL_NUMERIC ^c	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
VARBINARY	SQL_VARBINARY	SQL_C_BINARY
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA ^b	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR
VARGRAPHIC	SQL_WVARCHAR	SQL_C_DBCHAR
WCHAR	SQL_WCHAR	SQL_C_WCHAR
XML	SQL_XML	SQL_C_BINARY

Table 3. SQL symbolic and default data types (continued)

SQL data type	Symbolic SQL data type	Default symbolic C data type
<p>a LOB locator types are not persistent SQL data types, (columns can not be defined with a locator type, they are only used to describe parameter markers, or to represent a LOB value).</p> <p>b LONG data types and FOR BIT DATA data types should be replaced by an appropriate LOB types whenever possible.</p> <p>c NUMERIC is a synonym for DECIMAL on DB2 Version 9.1 for z/OS® (DB2 for z/OS), DB2 Server for VSE & VM and DB2 Database for Linux, UNIX, and Windows.</p>		
<p>Note: The data types DATE, DECIMAL, DECFLOAT(16), DECFLOAT(34), NUMERIC, TIME, and TIMESTAMP cannot be transferred to their default C buffer types without a conversion.</p>		

Related concepts:

- “Data types and data conversion in CLI applications” on page 39
- “LOB locators in CLI applications” on page 98
- “XML data type” in *XML Guide*

Related reference:

- “C data types for CLI applications” on page 43
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*

C data types for CLI applications

The following table lists the generic type definitions for each symbolic C type that is used in CLI applications.

C symbolic data type

This column contains C symbolic names, defined as integer values. These values are used in various function arguments to identify the C data type shown in the last column.

C type

This column contains C defined types, defined in `sqlcli.h` using a C typedef statement. The values in this column should be used to declare all DB2 CLI related variables and arguments, in order to make the application more portable. Refer to Table 6 on page 46 for a list of additional symbolic data types used for function arguments.

Base C type

This column is shown for reference only. All variables and arguments should be defined using the symbolic types in the previous column since the base C type is platform dependent. Some of the values are C structures that are described in Table 5 on page 44.

Table 4. C data types

C symbolic data type	C type	Base C type
SQL_C_BINARY	SQLCHAR	unsigned char
SQL_C_BIT	SQLCHAR	unsigned char or char (Value 1 or 0)
SQL_C_BLOB_LOCATOR ^a	SQLINTEGER	32-bit integer
SQL_C_CLOB_LOCATOR ^a	SQLINTEGER	32-bit integer
SQL_C_CHAR	SQLCHAR	unsigned char
SQL_C_DATA LINK	SQLCHAR	unsigned char
SQL_C_DBCHAR	SQLDBCHAR	wchar_t
SQL_C_DBCLOB_LOCATOR	SQLINTEGER	32-bit integer
SQL_C_DECIMAL64	SQLDECIMAL64	see Table 5
SQL_C_DECIMAL128	SQLDECIMAL128	see Table 5
SQL_C_DOUBLE	SQLDOUBLE	double
SQL_C_FLOAT	SQLREAL	float
SQL_C_LONG	SQLINTEGER	32-bit integer
SQL_C_NUMERIC ^b	SQL_NUMERIC_STRUCT	see Table 5
SQL_C_SBIGINT	SQLBIGINT	64-bit integer
SQL_C_SHORT	SQLSMALLINT	16-bit integer
SQL_C_TINYINT	SQLSCHAR	signed char (Range -128 to 127)
SQL_C_TYPE_DATE	DATE_STRUCT	see Table 5
SQL_C_TYPE_TIME	TIME_STRUCT	see Table 5
SQL_C_TYPE_TIMESTAMP	TIMESTAMP_STRUCT	see Table 5
SQL_C_UBIGINT	SQLUBIGINT	unsigned 64-bit integer
SQL_C_ULONG	SQLUINTEGER	unsigned 32-bit integer
SQL_C_USHORT	SQLUSMALLINT	unsigned 16-bit integer
SQL_C_UTINYINT	SQLUCHAR	unsigned char
SQL_C_WCHAR	SQLWCHAR	wchar_t

- **a** LOB Locator Types.
- **b** Windows only.

Note: SQL file reference data types (used in embedded SQL) are not needed in DB2 CLI.

Table 5. C structures

C type	Generic structure	Windows structure
DATE_STRUCT	<pre>typedef struct DATE_STRUCT { SQLSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; } DATE_STRUCT;</pre>	<pre>typedef struct tagDATE_STRUCT { SWORD year; UWORD month; UWORD day; } DATE_STRUCT;</pre>

Table 5. C structures (continued)

C type	Generic structure	Windows structure
TIME_STRUCT	<pre>typedef struct TIME_STRUCT { SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; } TIME_STRUCT;</pre>	<pre>typedef struct tagTIME_STRUCT { UWORD hour; UWORD minute; UWORD second; } TIME_STRUCT;</pre>
TIMESTAMP_STRUCT	<pre>typedef struct TIMESTAMP_STRUCT { SQLUSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; SQLINTEGER fraction; } TIMESTAMP_STRUCT;</pre>	<pre>typedef struct tagTIMESTAMP_STRUCT { SWORD year; UWORD month; UWORD day; UWORD hour; UWORD minute; UWORD second; UDWORD fraction; } TIMESTAMP_STRUCT;</pre>
SQLDECIMAL64	<pre>typedef struct tagSQLDECIMAL64 { union { SQLDOUBLE dummy; SQLCHAR dec64 [SQL_DECFLOAT16_ COEFFICIENT_LEN]; } udec64; } SQLDECIMAL64;</pre>	(No Windows structure. Only a generic structure.)
SQLDECIMAL128	<pre>typedef struct tagSQLDECIMAL128 { union { SQLDOUBLE dummy; SQLCHAR dec128 [SQL_DECFLOAT34_ COEFFICIENT_LEN]; } udec128; } SQLDECIMAL128;</pre>	(No Windows structure. Only a generic structure.)
SQL_NUMERIC_STRUCT	(No generic structure. Only a Windows structure.)	<pre>typedef struct tagSQL_NUMERIC_STRUCT { SQLCHAR precision; SQLCHAR scale; SQLCHAR sign; ^a SQLCHAR val[SQL_MAX_NUMERIC_LEN]; ^{b c} } SQL_NUMERIC_STRUCT;</pre>

Table 5. C structures (continued)

C type	Generic structure	Windows structure
Refer to Table 6 for more information on the SQLUSMALLINT C data type.		
<p>a Sign field: 1 = positive, 2 = negative</p> <p>b A number is stored in the val field of the SQL_NUMERIC_STRUCT structure as a scaled integer, in little endian mode (the leftmost byte being the least-significant byte). For example, the number 10.001 base 10, with a scale of 4, is scaled to an integer of 100010. Because this is 186AA in hexadecimal format, the value in SQL_NUMERIC_STRUCT would be "AA 86 01 00 00 ... 00", with the number of bytes defined by the SQL_MAX_NUMERIC_LEN #define.</p> <p>c The precision and scale fields of the SQL_C_NUMERIC data type are never used for input from an application, only for output from the driver to the application. When the driver writes a numeric value into the SQL_NUMERIC_STRUCT, it will use its own default as the value for the precision field, and it will use the value in the SQL_DESC_SCALE field of the application descriptor (which defaults to 0) for the scale field. An application can provide its own values for precision and scale by setting the SQL_DESC_PRECISION and SQL_DESC_SCALE fields of the application descriptor.</p>		

As well as the data types that map to SQL data types, there are also C symbolic types used for other function arguments such as pointers and handles. Both the generic and ODBC data types are shown below.

Note: There are two kinds of drivers that ship with the product: the DB2 CLI driver, and the 64-bit ODBC driver. The 64-bit ODBC Driver handles the differences with type definitions between various ODBC Managers.

Table 6. C Data types and base C data types

Defined C type	Base C type	Typical usage
SQLPOINTER	void *	Pointer to storage for data and parameters.
SQLHANDLE	<ol style="list-style-type: none"> 1. void * 2. 32-bit integer 	<p>Handle used to reference all 4 types of handle information.</p> <ol style="list-style-type: none"> 1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver 2. 32-bit value for all 32-bit platforms and 64-bit DB2 CLI Drivers
SQLHENV	<ol style="list-style-type: none"> 1. void * 2. 32-bit integer 	<p>Handle referencing environment information.</p> <ol style="list-style-type: none"> 1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver 2. 32-bit value for all 32-bit platforms and 64-bit DB2 CLI Drivers
SQLHDBC	<ol style="list-style-type: none"> 1. void * 2. 32-bit integer 	<p>Handle referencing database connection information.</p> <ol style="list-style-type: none"> 1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver 2. 32-bit value for all 32-bit platforms and 64-bit DB2 CLI Drivers
SQLHSTMT	<ol style="list-style-type: none"> 1. void * 2. 32-bit integer 	<p>Handle referencing statement information.</p> <ol style="list-style-type: none"> 1. 64-bit value for Windows 64-bit ODBC Driver and UNIX 64-bit ODBC Driver 2. 32-bit value for all 32-bit platforms and 64-bit DB2 CLI Drivers
SQLUSMALLINT	unsigned 16-bit integer	Function input argument for unsigned short integer values.

Table 6. C Data types and base C data types (continued)

Defined C type	Base C type	Typical usage
SQLINTEGER	unsigned 32-bit integer	Function input argument for unsigned long integer values.
SQLRETURN	16-bit integer	Return code from DB2 CLI functions.
SQLULEN	<ol style="list-style-type: none"> 1. unsigned 64-bit integer 2. unsigned 32-bit integer 	<ol style="list-style-type: none"> 1. Function input or output argument for unsigned 64-bit integer values (Windows 64-bit ODBC driver). 2. Function input or output argument for unsigned 32-bit integer values (all other drivers).
SQLLEN	<ol style="list-style-type: none"> 1. 64-bit integer 2. 32-bit integer 	<ol style="list-style-type: none"> 1. Function input or output argument for 64-bit integer values (Windows 64-bit ODBC driver). 2. Function input or output argument for 32-bit integer values (all other drivers).
SQLSETPOSROW	<ol style="list-style-type: none"> 1. 64-bit integer 2. 16-bit integer 	<ol style="list-style-type: none"> 1. Function input or output argument for 64-bit integer values (Windows 64-bit ODBC driver). 2. Function input or output argument for 16-bit integer values (all other drivers).

Related concepts:

- “Data types and data conversion in CLI applications” on page 39
- “LOB locators in CLI applications” on page 98

Related reference:

- “SQL symbolic and default data types for CLI applications” on page 41

String handling in CLI applications

The following conventions deal with the various aspects of string arguments in DB2 CLI functions.

Length of string arguments:

Input strings can have an associated length argument which indicates either the exact length of the string (not including the null terminator), the special value `SQL_NTS` to indicate a null-terminated string, or `SQL_NULL_DATA` to pass a NULL value. If the length is set to `SQL_NTS`, DB2 CLI will determine the length of the string by locating the null terminator.

Output strings have two associated length arguments: an input length argument to specify the length of the allocated output buffer, and an output length argument to return the actual length of the string returned by DB2 CLI. The returned length value is the total length of the string available for return, regardless of whether it fits in the buffer or not.

For SQL column data, if the output is a null value, `SQL_NULL_DATA` is returned in the length argument and the output buffer is untouched. The descriptor field `SQL_DESC_INDICATOR_PTR` is set to `SQL_NULL_DATA` if the column value is a null value. For more information, including which other fields are set, see the descriptor `FieldIdentifier` argument values.

If a function is called with a null pointer for an output length argument, DB2 CLI will not return a length. When the output data is a NULL value, DB2 CLI cannot indicate that the value is NULL. If it is possible that a column in a result set can contain a NULL value, a valid pointer to the output length argument must always be provided. It is highly recommended that a valid output length argument always be used.

Performance hint:

If the length argument (*StrLen_or_IndPtr*) and the output buffer (*TargetValuePtr*) are contiguous in memory, DB2 CLI can return both values more efficiently, improving application performance. For example, if the following structure is defined:

```
struct
{
    SQLINTEGER pcbValue;
    SQLCHAR   rgbValue [BUFFER_SIZE];
} buffer;
```

and `&buffer.pcbValue` and `buffer.rgbValue` is passed to `SQLBindCol()`, DB2 CLI would update both values in one operation.

Null-termination of strings:

By default, every character string that DB2 CLI returns is terminated with a null terminator (hex 00), except for strings returned from graphic and DBCLOB data types into `SQL_C_CHAR` application variables. Graphic and DBCLOB data types that are retrieved into `SQL_C_DBCHAR` application variables are null terminated with a double byte null terminator. Also, string data retrieved into `SQL_C_WCHAR` are terminated with the Unicode null terminator 0x0000. This requires that all buffers allocate enough space for the maximum number of bytes expected, plus the null terminator.

It is also possible to use `SQLSetEnvAttr()` and set an environment attribute to disable null termination of variable length output (character string) data. In this case, the application allocates a buffer exactly as long as the longest string it expects. The application must provide a valid pointer to storage for the output length argument so that DB2 CLI can indicate the actual length of data returned; otherwise, the application will not have any means to determine this. The DB2 CLI default is to always write the null terminator.

It is possible, using the Patch1 CLI/ODBC configuration keyword, to force DB2 CLI to null terminate graphic and DBCLOB strings.

String truncation:

If an output string does not fit into a buffer, DB2 CLI will truncate the string to the size of the buffer, and write the null terminator. If truncation occurs, the function will return `SQL_SUCCESS_WITH_INFO` and an `SQLSTATE` of `01004` indicating truncation. The application can then compare the buffer length to the output length to determine which string was truncated.

For example, if `SQLFetch()` returns `SQL_SUCCESS_WITH_INFO`, and an `SQLSTATE` of `01004`, it means at least one of the buffers bound to a column is too small to hold the data. For each buffer that is bound to a column, the application can compare the buffer length with the output length and determine which column was truncated. You can also call `SQLGetDiagField()` to find out which column failed.

Interpretation of strings:

Normally, DB2 CLI interprets string arguments in a case-sensitive manner and does not trim any spaces from the values. The one exception is the cursor name input argument on the `SQLSetCursorName()` function: if the cursor name is not delimited (enclosed by double quotes) the leading and trailing blanks are removed and case is ignored.

Blank padding of strings:

DB2 UDB Version 8.1.4 and later do not pad strings with blanks to fit the column size, as was the behavior in releases of DB2 UDB from Version 8.1 through to Version 8.1.4. With DB2 UDB Version 8.1.4 and later, a string may have a length which differs from the length defined for the CHAR column if code page conversion occurred. For releases of DB2 UDB before Version 8.1.4, strings would be padded with blanks to fill the column size; these blanks would be returned as part of the string data when the string was fetched from the CHAR column.

Related concepts:

- “SQLSTATES for DB2 CLI” on page 51

Related reference:

- “C data types for CLI applications” on page 43
- “CLI function return codes” on page 50
- “Descriptor header and record field initialization values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “Patch1 CLI/ODBC configuration keyword” on page 350
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetEnvAttr function (CLI) - Set environment attribute” in *Call Level Interface Guide and Reference, Volume 2*

Diagnostics in CLI applications overview

Diagnostics refers to dealing with warning or error conditions generated within an application. There are two levels of diagnostics returned when calling DB2 CLI functions:

- return codes
- detailed diagnostics (SQLSTATES, messages, SQLCA)

Each CLI function returns the function return code as a basic diagnostic. Both `SQLGetDiagRec()` and `SQLGetDiagField()` provide more detailed diagnostic information. If the diagnostic originates at the DBMS, the `SQLGetSQLCA()` function provides access to the SQLCA. This arrangement lets applications handle the basic flow control based on return codes, and use the SQLSTATES along with the SQLCA to determine the specific causes of failure and to perform specific error handling.

Both `SQLGetDiagRec()` and `SQLGetDiagField()` return three pieces of information:

- SQLSTATE

- Native error: if the diagnostic is detected by the data source, this is the SQLCODE; otherwise, this is set to -99999.
- Message text: this is the message text associated with the SQLSTATE.

SQLGetSQLCA() returns the SQLCA for access to specific fields, but should only be used when SQLGetDiagRec() or SQLGetDiagField() cannot provide the desired information.

Related concepts:

- “SQLSTATES for DB2 CLI” on page 51

Related reference:

- “CLI function return codes” on page 50
- “SQLCA (SQL communications area)” in *SQL Reference, Volume 1*
- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetSQLCA function (CLI) - Get SQLCA data structure” in *Call Level Interface Guide and Reference, Volume 2*

CLI function return codes

The following table lists all possible return codes for DB2 CLI functions.

Table 7. DB2 CLI Function return codes

Return code	Explanation
SQL_SUCCESS	The function completed successfully, no additional SQLSTATE information is available.
SQL_SUCCESS_WITH_INFO	The function completed successfully with a warning or other information. Call SQLGetDiagRec() or SQLGetDiagField() to receive the SQLSTATE and any other informational messages or warnings. The SQLSTATE will have a class of '01'.
SQL_STILL_EXECUTING	The function is running asynchronously and has not yet completed. The DB2 CLI driver has returned control to the application after calling the function, but the function has not yet finished executing.
SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. When this is returned after the execution of an SQL statement, additional information may be available and can be obtained by calling SQLGetDiagRec() or SQLGetDiagField().
SQL_NEED_DATA	The application tried to execute an SQL statement but DB2 CLI lacks parameter data that the application had indicated would be passed at execute time.
SQL_ERROR	The function failed. Call SQLGetDiagRec() or SQLGetDiagField() to receive the SQLSTATE and any other error information.
SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle). This is a programming error. No further information is available.

The following code segment from `tut_read.c` shows how a function return code, `SQL_NO_DATA_FOUND`, can be used to control when data retrieval should stop:

```
while (cliRC != SQL_NO_DATA_FOUND)
{
    printf("    %-8d %-14.14s \n", deptnumb.val, location.val);

    /* fetch next row */
    cliRC = SQLFetch(hstmt);
    STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
}
```

Related concepts:

- “Diagnostics in CLI applications overview” on page 49
- “Handles in CLI” on page 15

Related reference:

- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*

SQLSTATES for DB2 CLI

SQLSTATES are alphanumeric strings of 5 characters (bytes) with a format of `ccsss`, where `cc` indicates class and `sss` indicates subclass. Any SQLSTATE that has a class of:

- ‘01’, is a warning.
- ‘HY’, is generated by the DB2 CLI or ODBC driver.
- ‘IM’, is generated by the ODBC driver manager.

Note: Versions of DB2 CLI before Version 5 returned SQLSTATES with a class of ‘S1’ rather than ‘HY’. To force the CLI driver to return ‘S1’ SQLSTATES, the application should set the environment attribute `SQL_ATTR_ODBC_VERSION` to the value `SQL_OV_ODBC2`.

DB2 CLI SQLSTATES include both additional IBM defined SQLSTATES that are returned by the database server, and DB2 CLI defined SQLSTATES for conditions that are not defined in the ODBC version 3 and ISO SQL/CLI specifications. This allows for the maximum amount of diagnostic information to be returned. When running applications in an ODBC environment, it is also possible to receive ODBC defined SQLSTATES.

Follow these guidelines for using SQLSTATES within your application:

- Always check the function return code before calling `SQLGetDiagRec()` to determine if diagnostic information is available.
- Use the SQLSTATES rather than the native error code.
- To increase your application’s portability, only build dependencies on the subset of DB2 CLI SQLSTATES that are defined by the ODBC version 3 and ISO SQL/CLI specifications, and return the additional ones as information only. A dependency in an application is a logic flow decision based on specific SQLSTATES.

Note: It may be useful to build dependencies on the class (the first 2 characters) of the SQLSTATES.

- For maximum diagnostic information, return the text message along with the SQLSTATE (if applicable, the text message will also include the IBM defined SQLSTATE). It is also useful for the application to print out the name of the function that returned the error.
- Ensure that the string allocated for the SQLSTATE includes space for the null termination character returned by DB2 CLI.

The following code segment from `utilcli.c` shows how diagnostic information, such as SQLSTATES, can be retrieved and displayed:

```
void HandleDiagnosticsPrint(SQLSMALLINT htype, /* handle type identifier */
                           SQLHANDLE hndl /* handle */ )
{
    SQLCHAR message[SQL_MAX_MESSAGE_LENGTH + 1];
    SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
    SQLINTEGER sqlcode;
    SQLSMALLINT length, i;

    i = 1;

    /* get multiple field settings of diagnostic record */
    while (SQLGetDiagRec(htype,
                        hndl,
                        i,
                        sqlstate,
                        &sqlcode,
                        message,
                        SQL_MAX_MESSAGE_LENGTH + 1,
                        &length) == SQL_SUCCESS)
    {
        printf("\n SQLSTATE          = %s\n", sqlstate);
        printf(" Native Error Code = %ld\n", sqlcode);
        printf("%s\n", message);
        i++;
    }

    printf("-----\n");
}
```

You can use the CLI/ODBC trace facility to gain a better understanding of how your application calls DB2, including any errors that may occur.

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189

Related reference:

- “CLI function return codes” on page 50
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “utilcli.c -- Utility functions used by DB2 CLI samples”

Termination

Terminating a CLI application

After you have initialized your CLI application and processed transactions, you must terminate the application to properly disconnect from the data source and free resources.

Prerequisites:

Before terminating your application, you should have initialized your CLI application and completed processing of all transactions.

Procedure:

To terminate a CLI application:

1. Disconnect from the data source by calling `SQLDisconnect()`.
2. Free the connection handle by calling `SQLFreeHandle()` with a *HandleType* argument of `SQL_HANDLE_DBC`.

If multiple database connections exist, repeat steps 1 - 2 until all connections are closed and connection handles freed.

3. Free the environment handle by calling `SQLFreeHandle()` with a *HandleType* argument of `SQL_HANDLE_ENV`.

Related concepts:

- “Transaction processing in CLI overview” on page 20

Related tasks:

- “Freeing statement resources in CLI applications” on page 37
- “Initializing CLI applications” on page 18

Related reference:

- “SQLDisconnect function (CLI) - Disconnect from a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbconn.c -- How to connect to and disconnect from a database”
- “dbmcon.c -- How to use multiple databases”
- “utilcli.c -- Utility functions used by DB2 CLI samples”

Chapter 4. Programming hints and tips

Programming hints and tips for CLI applications

This topic discusses the following subjects:

- “KEEPDYNAMIC support”
- “Common connection attributes” on page 56
- “Common statement attributes” on page 56
- “Reusing statement handles” on page 57
- “Binding and SQLGetData()” on page 57
- “Limiting use of catalog functions” on page 57
- “Column names of function generated result sets” on page 58
- “DB2 CLI-specific functions loaded from ODBC applications” on page 58
- “Global dynamic statement caching” on page 58
- “Data insertion and retrieval optimization” on page 58
- “Large object data optimization” on page 58
- “Case sensitivity of object identifiers” on page 59
- “SQLDriverConnect() versus SQLConnect()” on page 59
- “Turning off statement scanning” on page 59
- “Holding cursors across rollbacks” on page 59
- “Preparing compound SQL sub-statements” on page 60
- “User-defined types casting” on page 60
- “Deferred prepare to reduce network flow” on page 61

KEEPDYNAMIC behavior refers to the server’s ability to keep a dynamic statement in a prepared state, even after a commit has been performed. This behavior eliminates the need for the client to prepare the statement again, the next time the statement is executed. Some DB2 CLI/ODBC applications on the client might improve their performance by taking advantage of the KEEPDYNAMIC behavior on servers that are DB2 for z/OS and OS/390® Version 7 and later.

Complete the following steps to enable KEEPDYNAMIC behavior:

1. Enable the dynamic statement cache on the DB2 for z/OS and OS/390 server (refer to the DB2 for z/OS and OS/390 server documentation).
2. Bind the db2clipk.bnd file on your DB2 Database for Linux, UNIX, and Windows client with the KEEPDYNAMIC and COLLECTION options. The following example shows how to bind db2clipk.bnd, creating a collection named KEEPDYNC:
 - db2 connect to *database_name* user *userid* using *password*
 - db2 bind db2clipk.bnd SQLERROR CONTINUE BLOCKING ALL KEEPDYNAMIC YES COLLECTION KEEPDYNC GRANT PUBLIC
 - db2 connect reset
3. Inform the client that the KEEPDYNAMIC bind option is enabled for your collection by performing either of the following:
 - Set the following CLI/ODBC configuration keywords in the db2cli.ini file: KeepDynamic = 1, CurrentPackageSet = collection name created in Step 2. For example:

```
[dbname]
KeepDynamic=1
CurrentPackageSet=KEEPDYNC
```

- Set the SQL_ATTR_KEEPPDYNAMIC and SQL_ATTR_CURRENT_PACKAGE_SET connection attributes in the DB2 CLI/ODBC application. For example:

```
SQLSetConnectAttr(hDbc,
                  SQL_ATTR_KEEP_DYNAMIC,
                  (SQLPOINTER) 1,
                  SQL_IS_UINTEGER );

SQLSetConnectAttr(hDbc,
                  SQL_ATTR_CURRENT_PACKAGE_SET,
                  (SQLPOINTER) "KEEPDYNC",
                  SQL_NTS);
```

Refer to the DB2 for OS/390 and z/OS documentation for further information on KEEPPDYNAMIC behavior and configuration.

Common connection attributes:

The following connection attributes might need to be set by DB2 CLI applications:

- SQL_ATTR_AUTOCOMMIT - Generally this attribute should be set to SQL_AUTOCOMMIT_OFF, since each commit request can generate extra network flow. Only leave SQL_AUTOCOMMIT_ON on if specifically needed.

Note: The default is SQL_AUTOCOMMIT_ON.

- SQL_ATTR_TXN_ISOLATION - This connection attribute determines the isolation level at which the connection or statement will operate. The isolation level determines the level of concurrency possible, and the level of locking required to execute the statement. Applications need to choose an isolation level that maximizes concurrency, yet ensures data consistency.

Common statement attributes:

The following statement attributes might need to be set by DB2 CLI applications:

- SQL_ATTR_MAX_ROWS - Setting this attribute limits the number of rows returned to the application from query operations. This can be used to avoid overwhelming an application with a very large result set generated inadvertently, which is especially useful for applications on clients with limited memory resources.

Setting SQL_ATTR_MAX_ROWS while connected to DB2 for z/OS and OS/390 Version 7 and later will add "OPTIMIZE FOR n ROWS" and "FETCH FIRST n ROWS ONLY" clauses to the statement. For versions of DB2 for OS/390 prior to Version 7 and any DBMS that does not support the "FETCH FIRST n ROWS ONLY" clause, the full result set is still generated at the server using the "OPTIMIZE FOR n ROWS" clause, however DB2 CLI will count the rows on the client and only fetch up to SQL_ATTR_MAX_ROWS rows.

- SQL_ATTR_CURSOR_HOLD - This statement attribute determines if the cursor for this statement will be declared by DB2 CLI using the WITH HOLD clause. Resources associated with statement handles can be better utilized by the server if the statements that do not require cursor-hold behavior have this attribute set to SQL_CURSOR_HOLD_OFF. The efficiency gains obtained by the proper use of this attribute are considerable on OS/390 and z/OS.

Note: Many ODBC applications expect a default behavior where the cursor position is maintained after a commit.

- `SQL_ATTR_TXN_ISOLATION` - DB2 CLI allows the isolation level to be set at the statement level, however, it is recommended that the isolation level be set at the connection level. The isolation level determines the level of concurrency possible, and the level of locking required to execute the statement.

Resources associated with statement handles can be better utilized by DB2 CLI if statements are set to the required isolation level, rather than leaving all statements at the default isolation level. This should only be attempted with a thorough understanding of the locking and isolation levels of the connected DBMS.

Applications should use the minimum isolation level possible to maximize concurrency.

Reusing statement handles:

Each time a CLI application declares a statement handle, the DB2 CLI driver allocates and then initializes an underlying data structure for that handle. To increase performance, CLI applications can reuse statement handles with different statements, thereby avoiding the costs associated with statement handle allocation and initialization.

Note: Before reusing statement handles, memory buffers and other resources used by the previous statement might need to be released by calling the `SQLFreeStmt()` function. Also, statement attributes previously set on a statement handle (for example, `SQL_ATTR_PARAMSET_SIZE`) need to be explicitly reset, otherwise they might be inherited by all future statements using the statement handle.

Binding and `SQLGetData()`:

Generally it is more efficient to bind application variables or file references to result sets than to use `SQLGetData()`. When the data is in a LOB column, LOB functions are preferable to `SQLGetData()` (see “Large object data optimization” on page 58 for more information). Use `SQLGetData()` when the data value is large variable-length data that:

- must be received in pieces, or
- might not need to be retrieved.

Limiting use of catalog functions:

Catalog functions, such as `SQLTables()`, force the DB2 CLI driver to query the DBMS catalog tables for information. The queries issued are complex and the DBMS catalog tables can be very large. In general, try to limit the number of times the catalog functions are called, and limit the number of rows returned.

The number of catalog function calls can be reduced by calling the function once, and having the application store (cache) the data.

The number of rows returned can be limited by specifying a:

- Schema name or pattern for all catalog functions
- Table name or pattern for all catalog functions other than `SQLTables()`
- Column name or pattern for catalog functions that return detailed column information.

Remember that although an application might be developed and tested against a data source with hundreds of tables, it might be run against a database with thousands of tables. Consider this likelihood when developing applications.

Close any open cursors (call `SQLCloseCursor()` or `SQLFreeStmt()` with `SQL_CLOSE Option`) for statement handles used for catalog queries to release any locks against the catalog tables. Outstanding locks on the catalog tables can prevent `CREATE`, `DROP` or `ALTER` statements from executing.

Column names of function generated result sets:

The column names of the result sets generated by catalog and information functions might change as the ODBC and CLI standards evolve. The *position* of the columns, however, will not change.

Any application dependency should be based on the column position (*iCol* parameter used in `SQLBindCol()`, `SQLGetData()`, and `SQLDescribeCol()`) and not the name.

DB2 CLI-specific functions loaded from ODBC applications:

The ODBC Driver Manager maintains its own set of statement handles which it maps to the CLI statement handles on each call. When a DB2 CLI function is called directly, it must be passed to the CLI driver statement handle, as the CLI driver does not have access to the ODBC mapping.

Call `SQLGetInfo()` with the `SQL_DRIVER_HSTMT` option to obtain the DB2 CLI statement handle (HSTMT). The DB2 CLI functions can then be called directly from the shared library or DLL, passing the HSTMT argument where required.

Global dynamic statement caching:

DB2 servers at version 5 or later for UNIX or Windows have a *global dynamic statement cache*. This cache is used to store the most popular access plans for prepared dynamic SQL statements.

Before each statement is prepared, the server automatically searches this cache to see if an access plan has already been created for this exact SQL statement (by this application or any other application or client). If so, the server does not need to generate a new access plan, but will use the one in the cache instead. There is now no need for the application to cache connections at the client unless connecting to a server that does not have a global dynamic statement cache.

Data insertion and retrieval optimization:

The methods that describe using arrays to bind parameters and retrieve data use compound SQL to optimize network flow. Use these methods as much as possible.

Large object data optimization:

Use LOB data types and the supporting functions for long strings whenever possible. Unlike `LONG VARCHAR`, `LONG VARBINARY`, and `LONG VARGRAPHIC` types, LOB data values can use LOB locators and functions such as `SQLGetPosition()` and `SQLGetSubString()` to manipulate large data values at the server.

LOB values can also be fetched directly to a file, and LOB parameter values can be read directly from a file. This saves the overhead of the application transferring data via application buffers.

Case sensitivity of object identifiers:

All database object identifiers, such as table names, view names and column names are stored in the catalog tables in uppercase unless the identifier is delimited. If an identifier is created using a delimited name, the exact case of the name is stored in the catalog tables.

When an identifier is referenced within an SQL statement, it is treated as case *insensitive* unless it is delimited.

For example, if the following two tables are created,

```
CREATE TABLE MyTable (id INTEGER)
CREATE TABLE "YourTable" (id INTEGER)
```

two tables will exist, MYTABLE and YourTable

Both of the following statements are equivalent:

```
SELECT * FROM MyTable (id INTEGER)
SELECT * FROM MYTABLE (id INTEGER)
```

The second statement below will fail with TABLE NOT FOUND since there is no table named YOURTABLE:

```
SELECT * FROM "YourTable" (id INTEGER) // executes without error
SELECT * FROM YourTable (id INTEGER) // error, table not found
```

All DB2 CLI catalog function arguments treat the names of objects as *case sensitive*, that is, as if each name was delimited.

SQLDriverConnect() versus SQLConnect():

Using SQLDriverConnect() allows the application to rely on the dialog box provided by DB2 CLI to prompt the user for the connection information.

If an application uses its own dialog boxes to query the connect information, the user should be able to specify additional connect options in the connection string. The string should also be stored and used as a default on subsequent connections.

Turning off statement scanning:

DB2 CLI by default, scans each SQL statement searching for vendor escape clause sequences.

If the application does not generate SQL statements that contain vendor escape clause sequences, then the SQL_ATTR_NOSCAN statement attribute should be set to SQL_NOSCAN_ON at the connection level so that DB2 CLI does not perform a scan for vendor escape clauses.

Holding cursors across rollbacks:

Applications that need to deal with complex transaction management issues might benefit from establishing multiple concurrent connections to the same database.

Each connection in DB2 CLI has its own transaction scope, so any actions performed on one connection do not affect the transactions of other connections.

For example, all open cursors within a transaction get closed if a problem causes the transaction to be rolled back. An application can use multiple connections to the same database to separate statements with open cursors; since the cursors are in separate transactions, a rollback on one statement does not affect the cursors of the other statements.

However, using multiple connections might mean bringing some data across to the client on one connection, and then sending it back to the server on the other connection. For example:

- Suppose in connection #1 you are accessing large object columns and have created LOB locators that map to portions of large object values.
- If in connection #2, you want to use (for example to insert) the portion of the LOB values represented by the LOB locators, you would have to move the LOB values in connection #1 first to the application, and then pass them to the tables that you are working with in connection #2. This is because connection #2 does not know anything about the LOB locators in connection #1.
- If you only had one connection, then you could just use the LOB locators directly. However, you would lose the LOB locators as soon as you rolled back your transaction.

Note: When multiple connections to a single database are used by an application, the application must be careful to synchronize access to database objects or it might experience various lock contention issues, as database locks are not shared between transactions. Updates by one connection can easily force other connections into a lock-wait state until the first connection releases the lock (through a COMMIT or ROLLBACK).

Preparing compound SQL sub-statements:

In order to maximize efficiency of the compound statement, sub-statements should be prepared before the BEGIN COMPOUND statement, and then executed within the compound statement.

This also simplifies error handling since prepare errors can be handled outside of the compound statement.

User-defined types and casting:

If a parameter marker is used in a predicate of a query statement, and the parameter is a user defined type, the statement must use a CAST function to cast either the parameter marker or the UDT.

For example, suppose the following type and table is defined:

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS

CREATE TABLE CUSTOMER (
    Cust_Num      CNUM NOT NULL,
    First_Name    CHAR(30) NOT NULL,
    Last_Name     CHAR(30) NOT NULL,
    Phone_Num     CHAR(20) WITH DEFAULT,
    PRIMARY KEY  (Cust_Num) )
```

Suppose also that the following SQL statement was then issued:

```
SELECT first_name, last_name, phone_num from customer
WHERE cust_num = ?
```

This statement would fail because the parameter marker cannot be of type CNUM and thus the comparison fails due to incompatible types.

Casting the column to integer (its base SQL type), allows the comparison to work since a parameter can be provided for type integer:

```
SELECT first_name, last_name, phone_num from customer
where cast( cust_num as integer ) = ?
```

Alternatively the parameter marker can be cast to INTEGER and the server can then apply the INTEGER to CNUM conversion:

```
SELECT first_name, last_name, phone_num FROM customer
where cust_num = cast( ? as integer )
```

Deferred prepare to reduce network flow:

In DB2 CLI, deferred prepare is on by default. The PREPARE request is not sent to the server until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance. This is of greatest benefit when an application generates queries with very small answer sets, because the overhead of requests and replies flowing over the network represents a large percentage of the processing time. In an environment where a DB2 Connect or DDCS gateway is used, there is a greater opportunity for cost reduction because four request and reply combinations are reduced to two.

Note: Functions such as `SQLDescribeParam()`, `SQLDescribeCol()`, `SQLNumParams()`, and `SQLNumResultCols()` require that the statement has been prepared. If the statement has not already been prepared, these functions trigger an immediate PREPARE request to the server, and the benefit of deferred prepare does not appear.

Related concepts:

- “Catalog functions for querying system catalog information in CLI applications” on page 167
- “Cursors in CLI applications” on page 65
- “Handles in CLI” on page 15
- “Large object usage in CLI applications” on page 96
- “Reduction of network flows with CLI array input chaining” on page 62
- “Vendor escape clauses in CLI applications” on page 171

Related tasks:

- “Executing compound SQL statements in CLI applications” on page 123
- “Retrieving array data in CLI applications using column-wise binding” on page 89
- “Retrieving array data in CLI applications using row-wise binding” on page 90

Related reference:

- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLConnect function (CLI) - Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Reduction of network flows with CLI array input chaining

CLI array input chaining is a feature that, when enabled, causes requests for the execution of prepared statements to be held and queued at the client until the chain is ended. Once the chain has been ended, all of the chained `SQLExecute()` requests at the client are then sent to the server in a single network flow.

The following sequence of events (presented as pseudocode) is an example of how CLI array input chaining can reduce the number of network flows to the server:

```
SQLPrepare (statement1)
SQLExecute (statement1)
SQLExecute (statement1)
/* the two execution requests for statement1 are sent to the server in
two network flows */

SQLPrepare (statement2)

/* enable chaining */
SQLSetStmtAttr (statement2, SQL_ATTR_CHAINING_BEGIN)

SQLExecute (statement2)
SQLExecute (statement2)
SQLExecute (statement2)

/* end chaining */
SQLSetStmtAttr (statement2, SQL_ATTR_CHAINING_END)

/* the three execution requests for statement2 are sent to the server
in a single network flow, instead of three separate flows */
```

If `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO` is returned when setting `SQL_ATTR_CHAINING_END`, then at least one statement in the chain of statements returned `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO` when it was executed. Use the CLI diagnostic functions `SQLGetDiagRec()` and `SQLGetDiagField()` to retrieve information about what has caused the error or warning.

Related concepts:

- “Programming hints and tips for CLI applications” on page 55

Related reference:

- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI function return codes” on page 50

Chapter 5. Cursors

Cursors	65	Retrieving data with scrollable cursors in a CLI application	76
Cursors in CLI applications	65	Bookmarks	78
Cursor considerations for CLI applications	68	Bookmarks in CLI applications	78
Result sets	70	Retrieving data with bookmarks in a CLI application	79
Result set terminology in CLI applications	70		
Rowset retrieval examples in CLI applications	71		
Specifying the rowset returned from the result set	73		

Cursors

Cursors in CLI applications

A CLI application uses a cursor to retrieve rows from a result set. A cursor is a moveable pointer to a row in the result table of an active query statement. With the introduction of the DB2 UDB Version 8 client, the support for an updatable scrollable cursor has been moved from the client to the server. This means that an application using the DB2 UDB client, Version 8 or higher, that requires updatable scrollable cursor needs to ensure that the server supports updatable scrollable cursors. DB2 UDB servers on Linux, UNIX and Windows, Version 8 and above, and DB2 for z/OS servers Version 7 and above support this feature. To access a scrollable cursor in a three-tier environment on DB2 for z/OS or DB2 for OS/390 Version 7 and above, the gateway must be running DB2 UDB Version 8 or above.

A cursor is opened when a dynamic SQL SELECT statement is successfully executed by `SQLExecute()` or `SQLExecDirect()`. There is typically a one-to-one correlation between application cursor operations and the operations performed by the DB2 CLI driver with the cursor. Immediately after the successful execution, the cursor is positioned before the first row of the result set, and `FETCH` operations through calls to `SQLFetch()`, `SQLFetchScroll()`, or `SQLExtendedFetch()` will advance the cursor one row at a time through the result set. When the cursor has reached the end of the result set, the next fetch operation will return `SQLCODE +100`. From the perspective of the CLI application, `SQLFetch()` returns `SQL_NO_DATA_FOUND` when the end of the result set is reached.

Types of cursors:

There are two types of cursors supported by DB2 CLI:

non-scrollable

Forward-only non-scrollable cursors are the default cursor type used by the DB2 CLI driver. This cursor type is unidirectional and requires the least amount of overhead processing.

scrollable

There are three types of scrollable cursors supported by DB2 CLI:

static This is a read-only cursor. Once it is created, no rows can be added or removed, and no values in any rows will change. The cursor is not affected by other applications accessing the same data. The isolation level of the statement used to create the cursor determines how the rows of the cursor are locked, if at all.

keyset-driven

Unlike a static scrollable cursor, a keyset-driven scrollable cursor can detect and make changes to the underlying data. Keyset cursors are based on row keys. When a keyset-driven cursor is first opened, it stores the keys in a keyset for the life of the entire result set. The keyset is used to determine the order and set of rows that are included in the cursor. As the cursor scrolls through the result set, it uses the keys in this keyset to retrieve the most recent values in the database, which are not necessarily the values that existed when the cursor was first opened. For this reason, changes are not reflected until the application scrolls to the row.

There are various types of changes to the underlying data that a keyset-driven cursor might or might not reflect:

- Changed values in existing rows. The cursor will reflect these types of changes. Because the cursor fetches a row from the database each time it is required, keyset-driven cursors always detect changes made by themselves and other cursors.
- Deleted rows. The cursor will reflect these types of changes. If a selected row in the rowset is deleted after the keyset is generated, it will appear as a "hole" in the cursor. When the cursor goes to fetch the row again from the database, it will realize that the row is no longer there.
- Added rows. The cursor will not reflect these types of changes. The set of rows is determined once, when the cursor is first opened. To see the inserted rows, the application must re-execute the query.

Note: DB2 CLI currently only supports keyset-driven cursors if the server supports them. The DB2 UDB Version 8 server now supports updatable scrollable cursors. This means that if applications require keyset cursor functionality and currently access DB2 for OS/390 Version 6 or DB2 for Unix and Windows Version 7 and earlier, the clients should not be migrated to DB2 UDB Version 8 or higher. The servers can be migrated to Version 8 or above.

dynamic

Dynamic scrollable cursors can detect all changes (inserts, deletes, and updates) to the result set, and make insertions, deletions and updates to the result set. Unlike keyset-driven cursors, dynamic cursors:

- detect rows inserted by other cursors
- omit deleted rows from the result set (keyset-driven cursors recognize deleted rows as "holes" in the result set)

Currently, dynamic scrollable cursors are only supported in DB2 CLI when accessing servers that are DB2 for z/OS Version 8.1 and later.

Cursor attributes:

The table below lists the default attributes for cursors in DB2 CLI.

Table 8. Default attributes for cursors in CLI

Cursor type	Cursor sensitivity	Cursor updatable	Cursor concurrency	Cursor scrollable
forward-only ^a	unspecified	non-updatable	read-only concurrency	non-scrollable
static	insensitive	non-updatable	read-only concurrency	scrollable
keyset-driven	sensitive	updatable	values concurrency	scrollable
dynamic ^b	sensitive	updatable	values concurrency	scrollable

a Forward-only is the default behavior for a scrollable cursor without the FOR UPDATE clause. Specifying FOR UPDATE on a forward-only cursor creates an updatable, lock concurrency, non-scrollable cursor.

b Values concurrency is the default behavior, however, DB2 on Linux, UNIX and Windows will also support lock concurrency, which will result with pessimistic locking.

Update of keyset-driven cursors:

A keyset-driven cursor is an updatable cursor. The CLI driver appends the FOR UPDATE clause to the query, except when the query is issued as a SELECT ... FOR READ ONLY query, or if the FOR UPDATE clause already exists. The default keyset-driven cursor is a values concurrency cursor. A values concurrency cursor results in optimistic locking, where locks are not held until an update or delete is attempted. If lock concurrency has been explicitly asked for, then pessimistic locking will be used and locks will be held as soon as the row is read. This level of locking is only supported against DB2 on Linux, UNIX and Windows servers. When an update or delete is attempted, the database server compares the previous values the application retrieved to the current values in the underlying table. If the values match, then the update or delete succeeds. If the values do not match, then the operation fails. If failure occurs, the application should query the values again and re-issue the update or delete if it is still applicable.

An application can update a keyset-driven cursor in two ways:

- Issue an UPDATE WHERE CURRENT OF <cursor name> or DELETE WHERE CURRENT OF <cursor name> using SQLPrepare() with SQLExecute() or SQLExecDirect()
- Use SQLSetPos() or SQLBulkOperations() to update, delete, or add a row to the result set.

Note: Rows added to a result set via SQLSetPos() or SQLBulkOperations() are inserted into the table on the server, but are not added to the server's result set. Therefore, these rows are not updatable nor are they sensitive to changes made by other transactions. The inserted rows will appear, however, to be part of the result set, since they are cached on the client. Any triggers that apply to the inserted rows will appear to the application as if they have not been applied. To make the inserted rows updatable, sensitive, and to see the result of applicable triggers, the application must issue the query again to regenerate the result set.

Related concepts:

- "Result set terminology in CLI applications" on page 70
- "Cursor considerations for CLI applications" on page 68

Related reference:

- “DELETE statement” in *SQL Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “UPDATE statement” in *SQL Reference, Volume 2*
- “CursorHold CLI/ODBC configuration keyword” on page 307
- “CursorTypes CLI/ODBC configuration keyword” on page 308

Cursor considerations for CLI applications

Which cursor type to use:

The first decision to make is between a forward-only cursor and a scrollable cursor. A forward-only cursor incurs less overhead than a scrollable cursor, and scrollable cursors have the potential for decreased concurrency. If your application does not need the additional features of a scrollable cursor, then you should use a non-scrollable cursor.

If a scrollable cursor is required then you have to decide between a static cursor, a keyset-driven cursor, or a dynamic cursor. A static cursor involves the least overhead. If the application does not need the additional features of a keyset-driven or dynamic cursor then a static cursor should be used.

Note: Currently, dynamic cursors are only supported when accessing servers that are DB2 for z/OS Version 8.1 and later.

If the application needs to detect changes to the underlying data or needs to add, update, or delete data from the cursor, then the application must use either a keyset-driven or dynamic cursor. To perform updates and deletions on rows in a dynamic scrollable cursor’s result set, the UPDATE or DELETE statement must include all the columns of at least one unique key in the base table. This can be the primary key or any other unique key. Because dynamic cursors incur more overhead and might have less concurrency than keyset-driven cursors, only choose dynamic cursors if the application needs to detect both changes made and rows inserted by other cursors.

If an application requests a scrollable cursor that can detect changes without specifying a particular cursor type, then DB2 CLI will assume that a dynamic cursor is not needed and provide a keyset-driven cursor. This behavior avoids the increased overhead and reduced concurrency that is incurred with dynamic cursors.

To determine the attributes of the types of cursors supported by the driver and DBMS, the application should call `SQLGetInfo()` with an *InfoType* of:

- `SQL_DYNAMIC_CURSOR_ATTRIBUTES1`

- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2

Unit of work considerations:

A cursor can be closed either explicitly or implicitly. An application can explicitly close a cursor by calling `SQLCloseCursor()`. Any further attempts to manipulate the cursor will result in error, unless the cursor is opened again. The implicit closure of a cursor depends on a several factors including how the cursor was declared and whether or not a COMMIT or ROLLBACK occurs.

By default, the DB2 CLI driver declares all cursors as WITH HOLD. This means that any open cursor will persist across COMMITs, thereby requiring the application to explicitly close each cursor. Be aware, however, that if a cursor is closed in autocommit mode, then any other open cursors that are not defined with the WITH HOLD option will be closed and all remaining open cursors will become unpositioned. (This means that no positioned updates or deletes can be performed without issuing another fetch.) There are two ways to change whether a cursor is declared WITH HOLD:

- Set the statement attribute `SQL_ATTR_CURSOR_HOLD` to `SQL_CURSOR_HOLD_ON` (default) or `SQL_CURSOR_HOLD_OFF`. This setting only affects cursors opened on the statement handle after this value has been set. It will not affect cursors already open.
- Set the CLI/ODBC configuration keyword `CursorHold` to change the default DB2 CLI driver behavior. Setting `CursorHold=1` preserves the default behavior of cursors declared as WITH HOLD, and `CursorHold=0` results in cursors being closed when each transaction is committed. You can override this keyword by setting the `SQL_ATTR_CURSOR_HOLD` statement attribute described above.

Note: A ROLLBACK will close all cursors, including those declared WITH HOLD.

Troubleshooting for applications created before scrollable cursor support:

Because scrollable cursor support is a newer feature, some CLI/ODBC applications that were working with previous releases of DB2 for OS/390 or DB2 for Linux, UNIX and Windows might encounter behavioral or performance changes. This occurs because before scrollable cursors were supported, applications that requested a scrollable cursor would receive a forward-only cursor. To restore an application’s previous behavior before scrollable cursor support, set the following configuration keywords in the `db2cli.ini` file:

Table 9. Configuration keyword values restoring application behavior before scrollable cursor support

Configuration keyword setting	Description
Patch2=6	Returns a message that scrollable cursors (keyset-driven, dynamic and static) are not supported. CLI automatically downgrades any request for a scrollable cursor to a forward-only cursor.

Table 9. Configuration keyword values restoring application behavior before scrollable cursor support (continued)

Configuration keyword setting	Description
DisableKeysetCursor=1	Disables keyset-driven scrollable cursors. This can be used to force the CLI driver to give the application a static cursor when a keyset-driven or dynamic cursor is requested.

Related concepts:

- “Commit modes in CLI applications” on page 30
- “Cursors in CLI applications” on page 65
- “Result set terminology in CLI applications” on page 70

Related reference:

- “COMMIT statement” in *SQL Reference, Volume 2*
- “ROLLBACK statement” in *SQL Reference, Volume 2*
- “SQLGetInfo function (CLI) - Get general information” in *Call Level Interface Guide and Reference, Volume 2*
- “CursorHold CLI/ODBC configuration keyword” on page 307
- “CursorTypes CLI/ODBC configuration keyword” on page 308
- “DisableKeysetCursor CLI/ODBC configuration keyword” on page 321
- “Patch2 CLI/ODBC configuration keyword” on page 352

Result sets

Result set terminology in CLI applications

The following terms describe result handling:

result set

The complete set of rows that satisfy the SQL SELECT statement. This is the set from which fetches retrieve rows to populate the rowset.

rowset

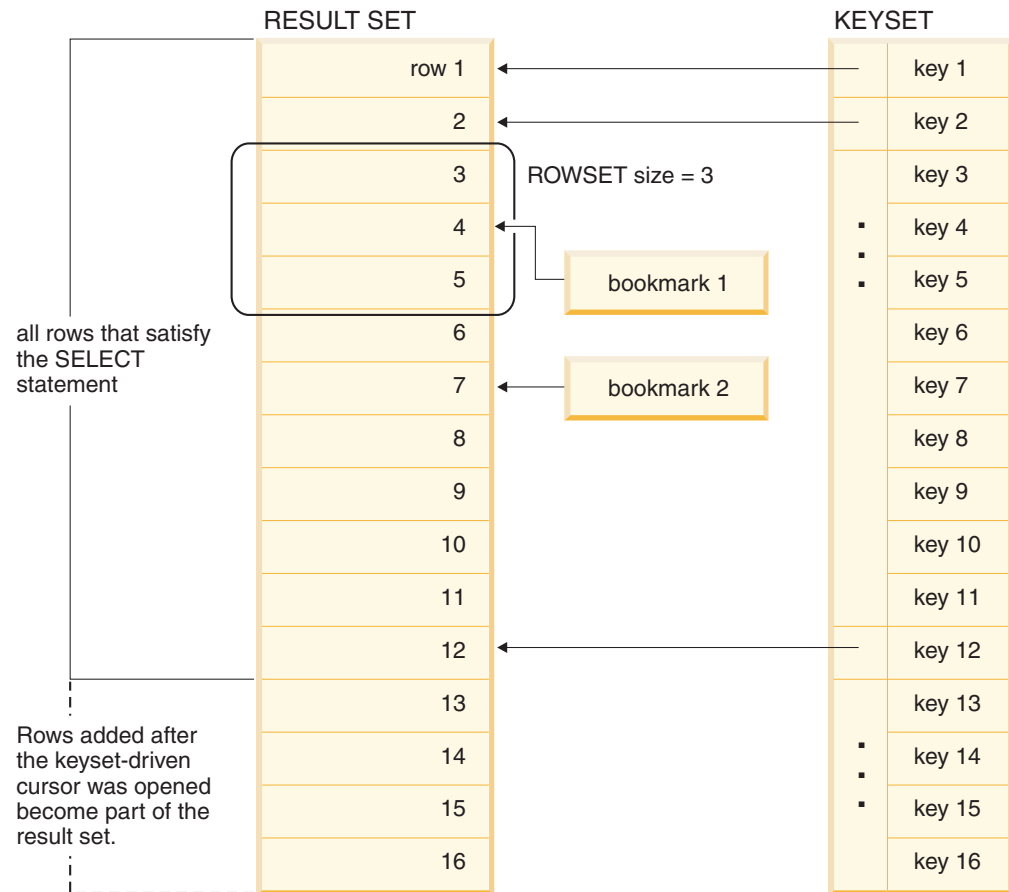
The subset of rows from the result set that is returned after each fetch. The application indicates the size of the rowset before the first fetch of data, and can modify the size before each subsequent fetch. Each call to `SQLFetch()`, `SQLFetchScroll()`, or `SQLExtendedFetch()` populates the rowset with the appropriate rows from the result set.

bookmark

It is possible to store a reference to a specific row in the result set called a bookmark. Once stored, the application can continue to move through the result set, then return to the bookmarked row to generate a rowset. You can also use a bookmark to perform updates and deletions with `SQLBulkOperations()`.

keyset A set of key values used to identify the set and order of rows that are included in a keyset-driven cursor. The keyset is created when a keyset-driven cursor is first opened. As the cursor scrolls through the result set, it uses the keys in the keyset to retrieve the current data values for each row.

The following figure demonstrates the relationship between the terms described above:



Related concepts:

- "Bookmarks in CLI applications" on page 78
- "Cursor considerations for CLI applications" on page 68
- "Cursors in CLI applications" on page 65

Related reference:

- "SELECT statement" in *SQL Reference, Volume 2*
- "SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows)" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLFetch function (CLI) - Fetch next row" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" in *Call Level Interface Guide and Reference, Volume 2*

Rowset retrieval examples in CLI applications

Partial rowset example:

When working with rowsets, you should verify what portion of the result set returned contains meaningful data. The application cannot assume that the entire

rowset will contain data. It must check the row status array after each rowset is created to determine the number of rows returned, because there are instances where the rowset will not contain a complete set of rows. For instance, consider the case where the rowset size is set to 10, and `SQLFetchScroll()` is called using `SQL_FETCH_ABSOLUTE` and `FetchOffset` is set to -3. This will attempt to return 10 rows starting 3 rows from the end of the result set. Only the first three rows of the rowset will contain meaningful data, however, and the application must ignore the rest of the rows.

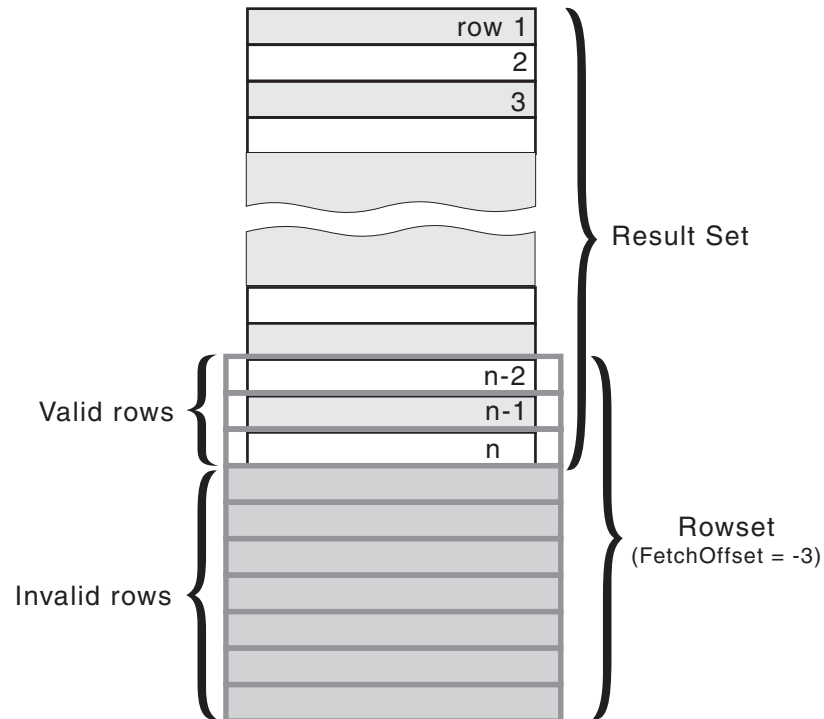


Figure 4. Partial rowset example

Fetch orientations example:

The following figure demonstrates a number of calls to `SQLFetchScroll()` using various `FetchOrientation` values. The result set includes all of the rows (from 1 to n), and the rowset size is 3. The order of the calls is indicated on the left, and the `FetchOrientation` values are indicated on the right.

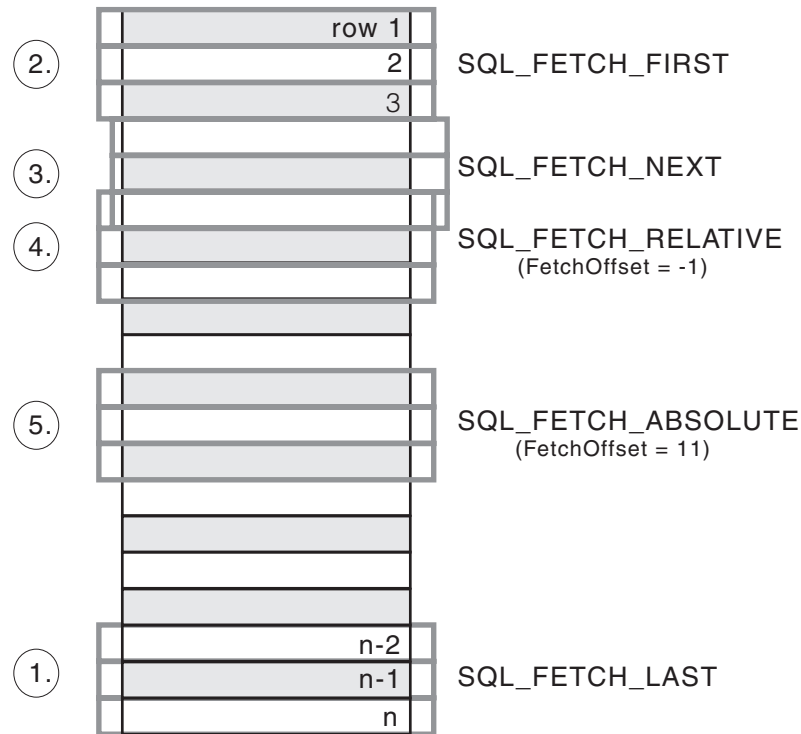


Figure 5. Example of retrieving rowsets

Related concepts:

- “Result set terminology in CLI applications” on page 70

Related reference:

- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*

Specifying the rowset returned from the result set

Before you begin to retrieve data, you need to establish the rowset that will be returned. This topic describes the steps associated with setting up the rowset.

Prerequisites:

Before specifying the rowset, ensure that you have initialized your CLI application.

Procedure:

DB2 CLI allows an application to specify a rowset for a non-scrollable or scrollable cursor that spans more than one row at a time. To effectively work with a rowset, an application should perform the following:

1. Specify the size of the rowset returned from calls to `SQLFetch()` or `SQLFetchScroll()` by setting the statement attribute `SQL_ATTR_ROW_ARRAY_SIZE` to the number of rows in the rowset. The default number of rows is 1. For example, to declare a rowset size of 35 rows, issue the following call:

```

#define ROWSET_SIZE 35
/* ... */
rc = SQLSetStmtAttr(hstmt,
                    SQL_ATTR_ROW_ARRAY_SIZE,
                    (SQLPOINTER) ROWSET_SIZE,
                    0);

```

2. Set up a variable that will store the number of rows returned. Declare a variable of type `SQLUINTEGER` and set the `SQL_ATTR_ROWS_FETCHED_PTR` statement attribute to point to this variable. In the following example, `rowsFetchedNb` will hold the number of rows returned in the rowset after each call to `SQLFetchScroll()`:

```

/* ... */

SQLUINTEGER rowsFetchedNb;

/* ... */

rc = SQLSetStmtAttr(hstmt,
                    SQL_ATTR_ROWS_FETCHED_PTR,
                    &rowsFetchedNb,
                    0);

```

3. Set up the row status array. Declare an array of type `SQLUSMALLINT` with the same number of rows as the size of the rowset (as determined in Step 1). Then specify the address of this array with the statement attribute `SQL_ATTR_ROW_STATUS_PTR`. For example:

```

/* ... */
SQLUSMALLINT row_status[ROWSET_SIZE];
/* ... */
/* Set a pointer to the array to use for the row status */
rc = SQLSetStmtAttr(
    hstmt,
    SQL_ATTR_ROW_STATUS_PTR,
    (SQLPOINTER) row_status,
    0);

```

The row status array provides additional information about each row in the rowset. After each call to `SQLFetch()` or `SQLFetchScroll()`, the array is updated. If the call to `SQLFetch()` or `SQLFetchScroll()` does not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, then the contents of the row status array are undefined. Otherwise, any of the row status array values will be returned (refer to the row status array section of the `SQLFetchScroll()` documentation for a complete list of values).

4. Position the rowset within the result set, indicating the position you want the rowset to begin. Specify this position by calling `SQLFetch()`, or `SQLFetchScroll()` with `FetchOrientation` and `FetchOffset` values. For example, the following call generates a rowset starting on the 11th row in the result set:

```

SQLFetchScroll(hstmt, /* Statement handle */
               SQL_FETCH_ABSOLUTE, /* FetchOrientation value */
               11); /* Offset value */

```

Scroll bar operations of a screen-based application can be mapped directly to the positioning of a rowset. By setting the rowset size to the number of lines displayed on the screen, the application can map the movement of the scroll bar to calls to `SQLFetchScroll()`.

Note: If the application can buffer data in the display and regenerate the result set to see updates, then use a forward-only cursor instead. This yields better performance for small result sets.

Rowset retrieved	FetchOrientation value	Scroll bar
First rowset	SQL_FETCH_FIRST	Home: Scroll bar at the top
Last rowset	SQL_FETCH_LAST	End: Scroll bar at the bottom
Next rowset	SQL_FETCH_NEXT (same as calling SQLFetch())	Page Down
Previous rowset	SQL_FETCH_PRIOR	Page Up
Rowset starting on next row	SQL_FETCH_RELATIVE with <i>FetchOffset</i> set to 1	Line Down
Rowset starting on previous row	SQL_FETCH_RELATIVE with <i>FetchOffset</i> set to -1	Line Up
Rowset starting on a specific row	SQL_FETCH_ABSOLUTE with <i>FetchOffset</i> set to an offset from the start (a positive value) or the end (a negative value) of the result set	Application generated
Rowset starting on a previously bookmarked row	SQL_FETCH_BOOKMARK with <i>FetchOffset</i> set to a positive or negative offset from the bookmarked row	Application generated

5. Check the rows fetched pointer after each rowset is created to determine the number of rows returned. Check the row status array for the status of each row, because there are instances where the rowset will not contain a complete set of rows. The application cannot assume that the entire rowset will contain data.

For instance, consider the case where the rowset size is set to 10, and `SQLFetchScroll()` is called using `SQL_FETCH_ABSOLUTE` and *FetchOffset* is set to -3. This will attempt to return 10 rows starting 3 rows from the end of the result set. Only the first three rows of the rowset will contain meaningful data, however, and the application must ignore the rest of the rows.

Related concepts:

- “Cursor considerations for CLI applications” on page 68
- “Result set terminology in CLI applications” on page 70

Related tasks:

- “Initializing CLI applications” on page 18
- “Retrieving data with bookmarks in a CLI application” on page 79
- “Retrieving data with scrollable cursors in a CLI application” on page 76

Related reference:

- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “C data types for CLI applications” on page 43
- “CLI function return codes” on page 50

Related samples:

- “tbread.c -- How to read data from tables”

Retrieving data with scrollable cursors in a CLI application

Scrollable cursors allow you to move throughout a result set. You can make use of this feature when retrieving data. This topic describes how to use scrollable cursors to retrieve data.

Prerequisites:

Before you retrieve data using scrollable cursors, ensure that you have initialized your CLI application.

Procedure:

To use scrollable cursors to retrieve data:

1. Specify the size of the rowset returned by setting the statement attribute `SQL_ATTR_ROW_ARRAY_SIZE` to the number of rows in the rowset. The default number of rows is 1. For example, to declare a rowset size of 35 rows, issue the following call:

```
#define ROWSET_SIZE 35
/* ... */
rc = SQLSetStmtAttr(hstmt,
                    SQL_ATTR_ROW_ARRAY_SIZE,
                    (SQLPOINTER) ROWSET_SIZE,
                    0);
```

2. Specify the type of scrollable cursor to use. Using `SQLSetStmtAttr()`, set the `SQL_ATTR_CURSOR_TYPE` statement attribute to `SQL_CURSOR_STATIC` for a static read-only cursor or to `SQL_CURSOR_KEYSET_DRIVEN` for a keyset-driven cursor. For example:

```
sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_CURSOR_TYPE,
                        (SQLPOINTER) SQL_CURSOR_STATIC,
                        0);
```

If the type of cursor is not set, the default forward-only non-scrollable cursor will be used.

3. Set up a variable that will store the number of rows returned. Declare a variable of type `SQLINTEGER` and set the `SQL_ATTR_ROWS_FETCHED_PTR` statement attribute to point to this variable. In the following example, *rowsFetchedNb* will hold the number of rows returned in the rowset after each call to `SQLFetchScroll()`:

```
/* ... */

SQLINTEGER rowsFetchedNb;

/* ... */

rc = SQLSetStmtAttr(hstmt,
                    SQL_ATTR_ROWS_FETCHED_PTR,
                    &rowsFetchedNb,
                    0);
```

4. Set up the row status array. Declare an array of type `SQLUSMALLINT` with the same number of rows as the size of the rowset (as determined in Step 1). Then specify the address of this array with the statement attribute `SQL_ATTR_ROW_STATUS_PTR`. For example:

```

/* ... */
SQLUSMALLINT   row_status[ROWSET_SIZE];
/* ... */
/* Set a pointer to the array to use for the row status */
rc = SQLSetStmtAttr(
    hstmt,
    SQL_ATTR_ROW_STATUS_PTR,
    (SQLPOINTER) row_status,
    0);

```

The row status array provides additional information about each row in the rowset. After each call to `SQLFetchScroll()`, the array is updated. If the call to `SQLFetchScroll()` does not return `SQL_SUCCESS` or `SQL_SUCCESS_WITH_INFO`, then the contents of the row status array are undefined. Otherwise, any of the row status array values will be returned (refer to the row status array section of the `SQLFetchScroll()` documentation for a complete list of values).

5. Optional: If you want to use bookmarks with the scrollable cursor, set the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE`. For example:

```

sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_USE_BOOKMARKS,
                        (SQLPOINTER) SQL_UB_VARIABLE,
                        0);

```

6. Issue an SQL SELECT statement.
7. Execute the SQL SELECT statement.
8. Bind the result set using either column-wise or row-wise binding.
9. Fetch a rowset of rows from the result set.
 - a. Call `SQLFetchScroll()` to fetch a rowset of data from the result set. Position the rowset within the result set indicating the position you want the rowset to begin. Specify this position by calling `SQLFetchScroll()` with *FetchOrientation* and *FetchOffset* values. For example, the following call generates a rowset starting on the 11th row in the result set:

```

SQLFetchScroll(hstmt,          /* Statement handle */
               SQL_FETCH_ABSOLUTE, /* FetchOrientation value */
               11);           /* Offset value */

```

- b. Check the row status array after each rowset is created to determine the number of rows returned, because there are instances where the rowset will not contain a complete set of rows. The application cannot assume that the entire rowset will contain data.

For instance, consider the case where the rowset size is set to 10, and `SQLFetchScroll()` is called using `SQL_FETCH_ABSOLUTE` and *FetchOffset* is set to -3. This will attempt to return 10 rows starting 3 rows from the end of the result set. Only the first three rows of the rowset will contain meaningful data, however, and the application must ignore the rest of the rows.

- c. Display or manipulate the data in the rows returned.
10. Close the cursor by calling `SQLCloseCursor()` or free the statement handle by calling `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_STMT`. Freeing the statement handles is not required every time retrieval has finished. The statement handles can be freed at a later time, when the application is freeing other handles.

Related concepts:

- “Bookmarks in CLI applications” on page 78

- “Cursor considerations for CLI applications” on page 68
- “Cursors in CLI applications” on page 65

Related tasks:

- “Initializing CLI applications” on page 18
- “Issuing SQL statements in CLI applications” on page 23
- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “SQLCloseCursor function (CLI) - Close cursor and discard pending results” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “C data types for CLI applications” on page 43
- “CLI function return codes” on page 50

Related samples:

- “tbread.c -- How to read data from tables”

Bookmarks

Bookmarks in CLI applications

When scrollable cursors are used, you can save a reference to any row in the result set using a bookmark. The application can then use that bookmark as a relative position to retrieve a rowset of information, or to update or delete a row when using keyset cursors. You can retrieve a rowset starting from the bookmarked row, or specify a positive or negative offset.

Once you have positioned the cursor to a row in a rowset using `SQLSetPos()`, you can obtain the bookmark value starting from column 0 using `SQLGetData()`. In most cases you will not want to bind column 0 and retrieve the bookmark value for every row, but use `SQLGetData()` to retrieve the bookmark value for the specific row you require.

A bookmark is only valid within the result set in which it was created. The bookmark value will be different if you select the same row from the same result set in two different cursors.

The only valid comparison is a byte-by-byte comparison between two bookmark values obtained from the same result set. If they are the same then they both point to the same row. Any other mathematical calculations or comparisons between bookmarks will not provide any useful information. This includes comparing bookmark values within a result set, and between result sets.

Related concepts:

- “Cursor considerations for CLI applications” on page 68
- “Result set terminology in CLI applications” on page 70

Related reference:

- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetPos function (CLI) - Set the cursor position in a rowset” in *Call Level Interface Guide and Reference, Volume 2*

Retrieving data with bookmarks in a CLI application

Bookmarks, available only when scrollable cursors are used, allow you to save a reference to any row in a result set. You can take advantage of this feature when retrieving data. This topic describes how to retrieve data using bookmarks.

Prerequisites:

Before you retrieve data with bookmarks, ensure that you have initialized your CLI application. The steps explained here should be performed in addition to those described in “Retrieving Data with Scrollable Cursors in a CLI Application”.

Procedure:

To use bookmarks with scrollable cursors to retrieve data:

1. Indicate that bookmarks will be used (if not already done so) by setting the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE. For example:

```
sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_USE_BOOKMARKS,
                        (SQLPOINTER) SQL_UB_VARIABLE,
                        0);
```

2. Get the bookmark value from the desired row in the rowset after executing the SELECT statement and retrieving the rowset using SQLFetchScroll(). Do this by calling SQLSetPos() to position the cursor within the rowset. Then call SQLGetData() to retrieve the bookmark value. For example:

```
sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 15);
/* ... */
sqlrc = SQLSetPos(hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
/* ... */
sqlrc = SQLGetData(hstmt, 0, SQL_C_LONG, bookmark.val, 4,
                  &bookmark.ind);
```

In most cases, you will not want to bind column 0 and retrieve the bookmark value for every row, but use SQLGetData() to retrieve the bookmark value for the specific row you require.

3. Store the bookmark location for the next call to SQLFetchScroll(). Set the SQL_ATTR_FETCH_BOOKMARK statement attribute to the variable that contains the bookmark value. For example, continuing from the example above, *bookmark.val* stores the bookmark value, so call SQLSetStmtAttr() as follows:

```
sqlrc = SQLSetStmtAttr(hstmt,
                        SQL_ATTR_FETCH_BOOKMARK_PTR,
                        (SQLPOINTER) bookmark.val,
                        0);
```

4. Retrieve a rowset based on the bookmark. Once the bookmark value is stored, the application can continue to use SQLFetchScroll() to retrieve data from the result set. The application can then move throughout the result set, but still retrieve a rowset based on the location of the bookmarked row at any point before the cursor is closed.

The following call to `SQLFetchScroll()` retrieves a rowset starting from the bookmarked row:

```
sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_BOOKMARK, 0);
```

The value 0 specifies the offset. You would specify -3 to begin the rowset 3 rows before the bookmarked row, or specify 4 to begin 4 rows after. For example, the following call from retrieves a rowset 4 rows after the bookmarked row:

```
sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_BOOKMARK, 4);
```

Note that the variable used to store the bookmark value is not specified in the `SQLFetchScroll()` call. It was set in the previous step using the statement attribute `SQL_ATTR_FETCH_BOOKMARK_PTR`.

Related concepts:

- “Bookmarks in CLI applications” on page 78
- “Cursor considerations for CLI applications” on page 68
- “Result set terminology in CLI applications” on page 70

Related tasks:

- “Initializing CLI applications” on page 18
- “Retrieving data with scrollable cursors in a CLI application” on page 76

Related reference:

- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetPos function (CLI) - Set the cursor position in a rowset” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “tbread.c -- How to read data from tables”

Chapter 6. Array input and output

Array input	81	Array output	85
Binding parameter markers in CLI applications with column-wise array input	81	Column binding in CLI applications	85
Binding parameter markers in CLI applications with row-wise array input	82	Result set retrieval into arrays in CLI applications	87
Parameter diagnostic information in CLI applications	83	Retrieving array data in CLI applications using column-wise binding	89
Changing parameter bindings in CLI applications with offsets	84	Retrieving array data in CLI applications using row-wise binding	90
		Changing column bindings in a CLI application with column binding offsets	91

Array input

Binding parameter markers in CLI applications with column-wise array input

To process an SQL statement that will be repeated with different values, you can use column-wise array input to achieve bulk inserts, deletes, or updates. This results in fewer network flows to the server because `SQLExecute()` does not have to be called repeatedly on the same SQL statement for each value. Column-wise array input allows arrays of storage locations to be bound to parameter markers. A different array is bound to each parameter.

Prerequisites:

Before binding parameter markers with column-wise binding, ensure that you have initialized your CLI application.

Restrictions:

For character and binary input data, the application uses the maximum input buffer size argument (*BufferLength*) of the `SQLBindParameter()` call to indicate to DB2 CLI the location of values in the input array. For other input data types, the length of each element in the array is assumed to be the size of the C data type.

Procedure:

To bind parameter markers using column-wise array input:

1. Specify the size of the arrays (the number rows to be inserted) by calling `SQLSetStmtAttr()` with the `SQL_ATTR_PARAMSET_SIZE` statement attribute.
2. Initialize and populate an array for each parameter marker to be bound.

Note: Each array must contain at least `SQL_ATTR_PARAMSET_SIZE` elements, otherwise, memory access violations may occur.

3. Optional: Indicate that column-wise binding is to be used by setting the `SQL_ATTR_BIND_TYPE` statement attribute to `SQL_PARAMETER_BIND_BY_COLUMN` (this is the default setting).
4. Bind each parameter marker to its corresponding array of input values by calling `SQLBindParameter()` for each parameter marker.

Related concepts:

- “Parameter diagnostic information in CLI applications” on page 83
- “Parameter marker binding in CLI applications” on page 26

Related tasks:

- “Initializing CLI applications” on page 18

Related reference:

- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Binding parameter markers in CLI applications with row-wise array input

To process an SQL statement that will be repeated with different values, you can use row-wise array input to achieve bulk inserts, deletes, or updates. This results in fewer network flows to the server because `SQLExecute()` does not have to be called repeatedly on the same SQL statement for each value. Row-wise array input allows an array of structures to be bound to parameters.

Prerequisites:

Before binding parameter markers with row-wise binding, ensure that you have initialized your CLI application.

Procedure:

To bind parameter markers using row-wise array input:

1. Initialize and populate an array of structures that contains two elements for each parameter: the first element contains the length/indicator buffer, and the second element holds the value itself. The size of the array corresponds to the number of values to be applied to each parameter. For example, the following array contains the length and value for three parameters:

```
struct { SQLINTEGER La; SQLINTEGER A; /* Information for parameter A */
        SQLINTEGER Lb; SQLCHAR B[4]; /* Information for parameter B */
        SQLINTEGER Lc; SQLCHAR C[11]; /* Information for parameter C */
    } R[n];
```

2. Indicate that row-wise binding is to be used by setting the `SQL_ATTR_PARAM_BIND_TYPE` statement attribute to the length of the struct created in the previous step, using `SQLSetStmtAttr()`.
3. Set the statement attribute `SQL_ATTR_PARAMSET_SIZE` to the number of rows of the array, using `SQLSetStmtAttr()`.
4. Bind each parameter to the first row of the array created in step 1 using `SQLBindParameter()`. For example,

```
/* Parameter A */
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
                     SQL_INTEGER, 5, 0, &R[0].A, 0, &R.La);
```

```
/* Parameter B */
```

```

rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
    10, 0, R[0].B, 10, &R.Lb);

/* Parameter C */
rc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
    3, 0, R[0].C, 3, &R.Lc);

```

Related concepts:

- “Parameter diagnostic information in CLI applications” on page 83
- “Parameter marker binding in CLI applications” on page 26

Related tasks:

- “Initializing CLI applications” on page 18

Related reference:

- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Parameter diagnostic information in CLI applications

A *parameter status array* is an array of one or more SQLSMALLINTs allocated by a CLI application. Each element in the array corresponds to an element in the input (or output) parameter array. If specified, the DB2 CLI driver updates the parameter status array with information about the processing status of each set of parameters included in an SQLExecute() or SQLExecDirect() call.

DB2 CLI updates the elements in the parameter status array with the following values:

- SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters.
- SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters, however, warning information is available in the diagnostics data structure.
- SQL_PARAM_ERROR: An error occurred in processing this set of parameters. Additional error information is available in the diagnostics data structure.
- SQL_PARAM_UNUSED: This parameter set was unused, possibly because a previous parameter set caused an error that aborted further processing.
- SQL_PARAM_DIAG_UNAVAILABLE: Diagnostic information is not available, possibly because an error was detected before the parameter set was even used (for example, an SQL statement syntax error).

A CLI application must call the SQLSetStmtAttr() function to set the SQL_ATTR_PARAM_STATUS_PTR attribute before DB2 CLI will update the parameter status array. Alternatively, the application can call the SQLSetDescField() function to set the SQL_DESC_ARRAY_STATUS_PTR field in the IPD descriptor to point to the parameter status array.

The statement attribute `SQL_ATTR_PARAMS_PROCESSED`, or the corresponding IPD descriptor header field `SQL_DESC_ROWS_PROCESSED_PTR`, can be used to return the number of sets of parameters that have been processed.

Once the application has determined what parameters had errors, it can use the statement attribute `SQL_ATTR_PARAM_OPERATION_PTR`, or the corresponding APD descriptor header field `SQL_DESC_ARRAY_STATUS_PTR`, (both of which point to an array of values) to control which sets of parameters are ignored in a second call to `SQLExecute()` or `SQLExecDirect()`.

Related tasks:

- “Binding parameter markers in CLI applications” on page 29

Related reference:

- “Descriptor FieldIdentifier argument values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescField function (CLI) - Set a single field of a descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Changing parameter bindings in CLI applications with offsets

When an application needs to change parameter bindings it can call `SQLBindParameter()` a second time. This will change the bound parameter buffer address and the corresponding length/indicator buffer address used. Instead of multiple calls to `SQLBindParameter()`, however, DB2 CLI also supports parameter binding offsets. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLExecute()` or `SQLExecDirect()`.

Prerequisites:

Before changing your parameter bindings, ensure that your application has been initialized.

Procedure:

To change parameter bindings by using offsets:

1. Call `SQLBindParameter()` as you had been to bind the parameters.
The first set of bound parameter buffer addresses and the corresponding length/indicator buffer addresses will act as a template. The application will then move this template to different memory locations using the offset.
2. Call `SQLExecute()` or `SQLExecDirect()` as you had been to execute the statement.
The values stored in the bound addresses will be used.
3. Initialize a variable to hold the memory offset value.

The statement attribute `SQL_ATTR_PARAM_BIND_OFFSET_PTR` points to the address of an `SQLINTEGER` buffer where the offset will be stored. This address must remain valid until the cursor is closed.

This extra level of indirection enables the use of a single memory variable to store the offset for multiple sets of parameter buffers on different statement handles. The application need only set this one memory variable and all of the offsets will be changed.

4. Store an offset value (number of bytes) in the memory location pointed to by the statement attribute set in the previous step.

The offset value is always added to the memory location of the originally bound values. This sum must point to a valid memory address.

5. Call `SQLExecute()` or `SQLExecDirect()` again. CLI will add the offset specified above to the locations used in the original call to `SQLBindParameter()` to determine where the parameters to be used are stored in memory.
6. Repeat steps 4 and 5 above as required.

Related concepts:

- “Cursors in CLI applications” on page 65

Related tasks:

- “Binding parameter markers in CLI applications with column-wise array input” on page 81
- “Binding parameter markers in CLI applications with row-wise array input” on page 82
- “Initializing CLI applications” on page 18
- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “C data types for CLI applications” on page 43
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Array output

Column binding in CLI applications

Columns may be bound to:

- Application storage

`SQLBindCol()` is used to bind application storage to the column. Data will be transferred from the server to the application at fetch time. Length of the available data to return is also set.

- LOB locators

`SQLBindCol()` is used to bind LOB locators to the column. Only the LOB locator (4 bytes) will be transferred from the server to the application at fetch time.

Once an application receives a locator it can be used in `SQLGetSubString()`, `SQLGetPosition()`, `SQLGetLength()`, or as the value of a parameter marker in another SQL statement. `SQLGetSubString()` can either return another locator, or the data itself. All locators remain valid until the end of the transaction in which they were created (even when the cursor moves to another row), or until it is freed using the `FREE LOCATOR` statement.

- Lob file references

`SQLBindFileToCol()` is used to bind a file to a LOB or XML column. DB2 CLI will write the data directly to a file, and update the *StringLength* and *IndicatorValue* buffers specified on `SQLBindFileToCol()`.

If the data value for the column is NULL and `SQLBindFileToCol()` was used, then *IndicatorValue* will be set to `SQL_NULL_DATA` and *StringLength* to 0.

The number of columns in a result set can be determined by calling `SQLNumResultCols()` or by calling `SQLColAttribute()` with the *DescType* argument set to `SQL_COLUMN_COUNT`.

The application can query the attributes (such as data type and length) of the column by first calling `SQLDescribeCol()` or `SQLColAttribute()`. This information can then be used to allocate a storage location of the correct data type and length, to indicate data conversion to another data type, or in the case of LOB data types, optionally return a locator.

An application can choose not to bind every column, or even not to bind any columns. Data in any of the columns can also be retrieved using `SQLGetData()` after the bound columns have been fetched for the current row. It is usually more efficient to bind application variables or file references to result sets than to use `SQLGetData()`. When the data is in a LOB column, LOB functions are preferable to `SQLGetData()`. Use `SQLGetData()` when the data value is large variable-length data that:

- must be received in pieces, or
- may not need to be retrieved.

Instead of multiple calls to `SQLBindCol()`, DB2 CLI also supports column binding offsets. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLFetch()` or `SQLFetchScroll()`. This can only be used with row wise binding, but will work whether the application retrieves a single row or multiple rows at a time.

When binding any variable length column, DB2 CLI will be able to write *StrLen_or_IndPtr* and *TargetValuePtr* in one operation if they are allocated contiguously. For example:

```
struct { SQLINTEGER StrLen_or_IndPtr;
        SQLCHAR TargetValuePtr[MAX_BUFFER];
        } column;
```

The most recent bind column function call determines the type of binding that is in effect.

Related concepts:

- “LOB locators in CLI applications” on page 98

Related tasks:

- “Changing column bindings in a CLI application with column binding offsets” on page 91

Related reference:

- “FREE LOCATOR statement” in *SQL Reference, Volume 2*
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*

Result set retrieval into arrays in CLI applications

One of the most common tasks performed by an application is to issue a query statement, and then fetch each row of the result set into application variables that have been bound using `SQLBindCol()`. If the application requires that each column or each row of the result set be stored in an array, each fetch must be followed by either a data copy operation or a new set of `SQLBindCol()` calls to assign new storage areas for the next fetch.

Alternatively, applications can eliminate the overhead of extra data copies or extra `SQLBindCol()` calls by retrieving multiple rows of data (called a rowset) at one time into an array.

Note: A third method of reducing overhead, which can be used on its own or with arrays, is to specify a binding offset. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLFetch()` or `SQLFetchScroll()`. This can only be used with row offset binding.

When retrieving a result set into an array, `SQLBindCol()` is also used to assign storage for application array variables. By default, the binding of rows is in column-wise fashion: this is similar to using `SQLBindParameter()` to bind arrays of input parameter values. Figure 6 on page 88 is a logical view of column-wise binding.

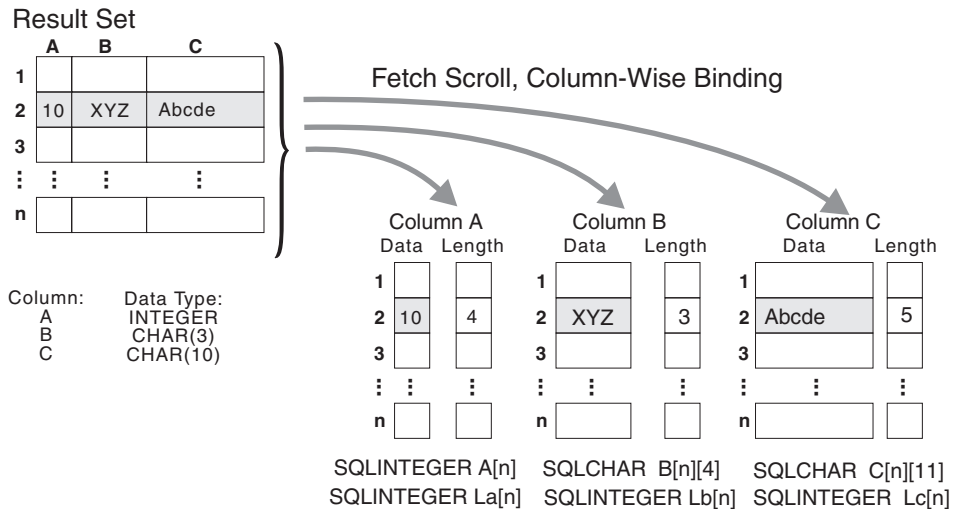


Figure 6. Column-wise binding

The application can also do row-wise binding which associates an entire row of the result set with a structure. In this case the rowset is retrieved into an array of structures, each of which holds the data in one row and the associated length fields. Figure 7 gives a pictorial view of row-wise binding.

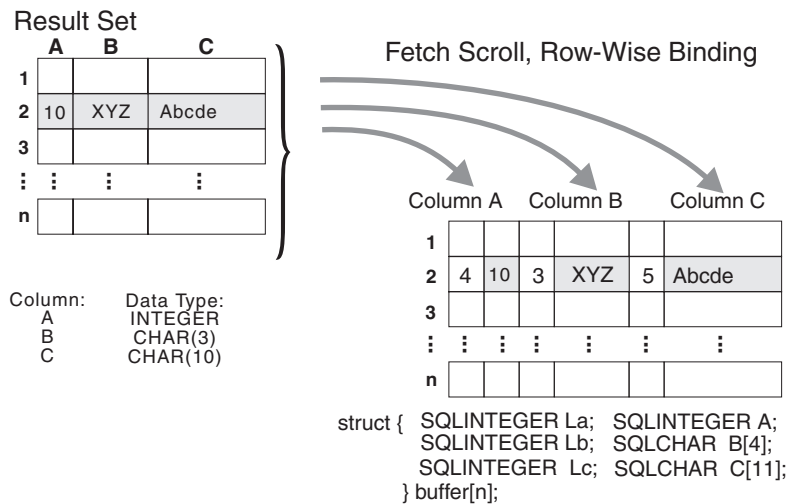


Figure 7. Row-wise binding

Related tasks:

- “Changing column bindings in a CLI application with column binding offsets” on page 91
- “Retrieving array data in CLI applications using column-wise binding” on page 89
- “Retrieving array data in CLI applications using row-wise binding” on page 90

Related reference:

- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*

Retrieving array data in CLI applications using column-wise binding

When retrieving data, you may want to retrieve more than one row at a time and store the data in an array. Instead of fetching and copying each row of data into an array, or binding to new storage areas, you can retrieve multiple rows of data at once using column-wise binding. Column-wise binding is the default row-binding method whereby each data value and its length is stored in an array.

Prerequisites:

Before using column-wise binding to retrieve data into arrays, ensure you have initialized your CLI application.

Procedure:

To retrieve data using column-wise binding:

1. Allocate an array of the appropriate data type for each column data value. This array will hold the retrieved data value.
2. Allocate an array of SQLINTEGER for each column. Each array will store the length of each column's data value.
3. Specify that column-wise array retrieval will be used by setting the `SQL_ATTR_ROW_BIND_TYPE` statement attribute to `SQL_BIND_BY_COLUMN` using `SQLSetStmtAttr()`.
4. Specify the number of rows that will be retrieved by setting the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute using `SQLSetStmtAttr()`.

When the value of the `SQL_ATTR_ROW_ARRAY_SIZE` attribute is greater than 1, DB2 CLI treats the deferred output data pointer and length pointer as pointers to arrays of data and length rather than to one single element of data and length of a result set column.

5. Prepare and execute the SQL statement used to retrieve the data.
6. Bind each array to its column by calling `SQLBindCol()` for each column.
7. Retrieve the data by calling `SQLFetch()` or `SQLFetchScroll()`.

When returning data, DB2 CLI uses the maximum buffer size argument (*BufferLength*) of `SQLBindCol()` to determine where to store successive rows of data in the array. The number of bytes available for return for each element is stored in the deferred length array. If the number of rows in the result set is greater than the `SQL_ATTR_ROW_ARRAY_SIZE` attribute value, multiple calls to `SQLFetchScroll()` are required to retrieve all the rows.

Related concepts:

- “Result set retrieval into arrays in CLI applications” on page 87

Related tasks:

- “Initializing CLI applications” on page 18
- “Preparing and executing SQL statements in CLI applications” on page 24
- “Retrieving array data in CLI applications using row-wise binding” on page 90

Related reference:

- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQL symbolic and default data types for CLI applications” on page 41

Related samples:

- “tbread.c -- How to read data from tables”

Retrieving array data in CLI applications using row-wise binding

When retrieving data, you may want to retrieve more than one row at a time and store the data in an array. Instead of fetching and copying each row of data into an array, or binding to new storage areas, you can retrieve multiple rows of data using row-wise binding. Row-wise binding associates an entire row of the result set with a structure. The rowset is retrieved into an array of structures, each of which holds the data in one row and the associated length fields.

Prerequisites:

Before using row-wise binding to retrieve data into arrays, ensure you have initialized your CLI application.

Procedure:

To retrieve data using row-wise binding:

1. Allocate an array of structures of size equal to the number of rows to be retrieved, where each element of the structure is composed of each row’s data value and each data value’s length.

For example, if each row of the result set consisted of Column A of type INTEGER, Column B of type CHAR(3), and Column C of type CHAR(10), then you would allocate the following structure, where n represents the number of rows in the result set:

```
struct {
    SQLINTEGER La; SQLINTEGER A;
    SQLINTEGER Lb; SQLCHAR B[4];
    SQLINTEGER Lc; SQLCHAR C[11];
} buffer[n];
```

2. Specify that row-wise array retrieval will be used by setting the SQL_ATTR_ROW_BIND_TYPE statement attribute, using SQLSetStmtAttr() to the size of the structure to which the result columns will be bound.
3. Specify the number of rows that will be retrieved by setting the SQL_ATTR_ROW_ARRAY_SIZE statement attribute using SQLSetStmtAttr().
4. Prepare and execute the SQL statement used to retrieve the data.
5. Bind each structure to the row by calling SQLBindCol() for each column of the row.

DB2 CLI treats the deferred output data pointer of SQLBindCol() as the address of the data field for the column in the first element of the array of structures. The deferred output length pointer is treated as the address of the associated length field of the column.

6. Retrieve the data by calling SQLFetchScroll().

When returning data, DB2 CLI uses the structure size provided with the `SQL_ATTR_ROW_BIND_TYPE` statement attribute to determine where to store successive rows in the array of structures.

Related concepts:

- “Result set retrieval into arrays in CLI applications” on page 87

Related tasks:

- “Initializing CLI applications” on page 18
- “Preparing and executing SQL statements in CLI applications” on page 24
- “Retrieving array data in CLI applications using column-wise binding” on page 89

Related reference:

- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “tbread.c -- How to read data from tables”

Changing column bindings in a CLI application with column binding offsets

When an application needs to change bindings (for a subsequent fetch, for example) it can call `SQLBindCol()` a second time. This will change the buffer address and length/indicator pointer used. Instead of multiple calls to `SQLBindCol()`, DB2 CLI supports column binding offsets. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLFetch()` or `SQLFetchScroll()`.

Prerequisites:

Before using column binding offsets to change result set bindings, ensure you have initialized your CLI application.

Restrictions:

This method can only be used with row-wise binding, but will work whether the application retrieves a single row or multiple rows at a time.

Procedure:

To change result set bindings using column binding offsets:

1. Call `SQLBindCol()` as usual to bind the result set. The first set of bound data buffer and length/indicator buffer addresses will act as a template. The application will then move this template to different memory locations using the offset.
2. Call `SQLFetch()` or `SQLFetchScroll()` as usual to fetch the data. The data returned will be stored in the locations bound above.
3. Set up a variable to hold the memory offset value.
 The statement attribute `SQL_ATTR_ROW_BIND_OFFSET_PTR` points to the address of an `SQLINTEGER` buffer where the offset will be stored. This address must remain valid until the cursor is closed.
 This extra level of indirection enables the use of a single memory variable to store the offset for multiple sets of bindings on different statement handles. The application need only set this one memory variable and all of the offsets will be changed.
4. Store an offset value (number of bytes) in the memory location pointed to by the statement attribute set in the previous step.
 The offset value is always added to the memory location of the originally bound values. This sum must point to a valid memory address with sufficient space to hold the next set of data.
5. Call `SQLFetch()` or `SQLFetchScroll()` again. CLI will add the offset specified above to the locations used in the original call to `SQLBindCol()`. This will determine where in memory to store the results.
6. Repeat steps 4 and 5 above as required.

Related concepts:

- “Column binding in CLI applications” on page 85

Related tasks:

- “Initializing CLI applications” on page 18
- “Retrieving array data in CLI applications using row-wise binding” on page 90

Related reference:

- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Chapter 7. Working with large amounts of data

Specifying parameter values at execute time for long data manipulation in CLI applications	93	Retrieving bulk data with bookmarks using SQLBulkOperations() in CLI applications	105
Data retrieval in pieces in CLI applications	95	Inserting bulk data with bookmarks using SQLBulkOperations() in CLI applications	106
Large object usage in CLI applications	96	Updating bulk data with bookmarks using SQLBulkOperations() in CLI applications	107
LOB locators in CLI applications	98	Deleting bulk data with bookmarks using SQLBulkOperations() in CLI applications	109
Fetching LOB data with LOB locators in CLI applications	99	Importing data with the CLI LOAD utility in CLI applications	110
Direct file input and output for LOB handling in CLI applications	101		
LOB usage in ODBC applications.	102		
Bulk data manipulation	103		
Long data for bulk inserts and updates in CLI applications	103		

Specifying parameter values at execute time for long data manipulation in CLI applications

When manipulating long data, it may not be feasible for the application to load the entire parameter data value into storage at the time the statement is executed, or when the data is fetched from the database. A method has been provided to allow the application to handle the data in a piecemeal fashion. The technique of sending long data in pieces is called *specifying parameter values at execute time*. It can also be used to specify values for fixed size non-character data types such as integers.

Prerequisites:

Before specifying parameter values at execute time, ensure you have initialized your CLI application.

Restrictions:

While the data-at-execution flow is in progress, the only DB2 CLI functions the application can call are:

- SQLParamData() and SQLPutData() as given in the sequence below.
- The SQLCancel() function which is used to cancel the flow and force an exit from the loops described below without executing the SQL statement.
- The SQLGetDiagRec() function.

Procedure:

A data-at-execute parameter is a bound parameter for which a value is prompted at execution time instead of stored in memory before SQLExecute() or SQLExecDirect() is called. To indicate such a parameter on an SQLBindParameter() call:

1. Set the input data length pointer to point to a variable that, at execute time, will contain the value SQL_DATA_AT_EXEC. For example:

```
/* dtlob.c */
/* ... */
SQLINTEGER blobInd ;
/* ... */
blobInd = SQL_DATA_AT_EXEC;
```

```

sqlrc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
                        SQL_BLOB, BUFSIZ, 0, (SQLPOINTER)inputParam,
                        BUFSIZ, &blobInd);

```

2. If there is more than one data-at-execute parameter, set each input data pointer argument to some value that it will recognize as uniquely identifying the field in question.
3. If there are any data-at-execute parameters when the application calls `SQLExecDirect()` or `SQLExecute()`, the call returns with `SQL_NEED_DATA` to prompt the application to supply values for these parameters. The application responds with the subsequent steps.
4. Call `SQLParamData()` to conceptually advance to the first such parameter. `SQLParamData()` returns `SQL_NEED_DATA` and provides the contents of the input data pointer argument specified on the associated `SQLBindParameter()` call to help identify the information required.
5. Pass the actual data for the parameter by calling `SQLPutData()`. Long data can be sent in pieces by calling `SQLPutData()` repeatedly.
6. Call `SQLParamData()` again after providing the entire data for this data-at-execute parameter.
7. If more data-at-execute parameters exist, `SQLParamData()` again returns `SQL_NEED_DATA` and the application repeats steps 4 and 5 above.

For example:

```

/* dtlob.c */
/* ... */
else
{
    sqlrc = SQLParamData( hstmt, (SQLPOINTER *) &valuePtr);
    /* ... */

    while ( sqlrc == SQL_NEED_DATA)
    {
        /*
         * if more than 1 parms used DATA_AT_EXEC then valuePtr would
         * have to be checked to determine which param needed data
         */
        while ( feof( pFile ) == 0 )
        {
            n = fread( buffer, sizeof(char), BUFSIZ, pFile);
            sqlrc = SQLPutData(hstmt, buffer, n);
            STMT_HANDLE_CHECK( hstmt, sqlrc);
            fileSize = fileSize + n;
            if ( fileSize > 102400u)
            {
                /* BLOB column defined as 100K MAX */
                /* ... */
                break;
            }
        }
        /* ... */
        sqlrc = SQLParamData( hstmt, (SQLPOINTER *) &valuePtr);
        /* ... */
    }
}

```

When all data-at-execute parameters have been assigned values, `SQLParamData()` completes execution of the SQL statement and returns a return value and diagnostics as the original `SQLExecDirect()` or `SQLExecute()` would have produced.

Related tasks:

- “Initializing CLI applications” on page 18

Related reference:

- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLCancel function (CLI) - Cancel statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLParamData function (CLI) - Get next parameter for which a data value is needed” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtlob.c -- How to read and write LOB data”

Data retrieval in pieces in CLI applications

Typically, an application might choose to allocate the maximum memory the column value could occupy and bind it via `SQLBindCol()`, based on information about a column in the result set (obtained via a call to `SQLDescribeCol()`, for example, or prior knowledge). However, in the case of character and binary data, the column can be arbitrarily long. If the length of the column value exceeds the length of the buffer the application can allocate or afford to allocate, a feature of `SQLGetData()` lets the application use repeated calls to obtain in sequence the value of a single column in more manageable pieces.

A call to `SQLGetData()` (which is called after `SQLFetch()`) returns `SQL_SUCCESS_WITH_INFO` (with `SQLSTATE 01004`) to indicate more data exists for this column. `SQLGetData()` is called repeatedly to get the remaining pieces of data until it returns `SQL_SUCCESS`, signifying that the entire data has been retrieved for this column.

For example:

```

/* dtlob.c */
/* ... */
sqlrc = SQLGetData(hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer,
                  BUFSIZ, &bufInd);

/* ... */
while( sqlrc == SQL_SUCCESS_WITH_INFO || sqlrc == SQL_SUCCESS )
{
  if ( bufInd > BUFSIZ) /* full buffer */
  {
    fwrite( buffer, sizeof(char), BUFSIZ, pFile);
  }
  else /* partial buffer on last GetData */
  {
    fwrite( buffer, sizeof(char), bufInd, pFile);
  }

  sqlrc = SQLGetData( hstmt, 1, SQL_C_BINARY, (SQLPOINTER)buffer,
                    BUFSIZ, &bufInd);
}
/* ... */

```

The function `SQLGetSubString()` can also be used to retrieve a specific portion of a large object value. For other alternative methods to retrieve long data, refer to the documentation on large object usage.

Related concepts:

- “Large object usage in CLI applications” on page 96

Related reference:

- “CLI function return codes” on page 50
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetSubString function (CLI) - Retrieve portion of a string value” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtlob.c -- How to read and write LOB data”

Large object usage in CLI applications

The term *large object* and the generic acronym *LOB* are used to refer to any type of large object. There are three LOB data types: Binary Large Object (BLOB), Character Large Object (CLOB), and Double-Byte Character Large Object (DBCLOB). These LOB data types are represented symbolically as SQL_BLOB, SQL_CLOB, SQL_DBCLOB respectively. The LOB symbolic constants can be specified or returned on any of the DB2 CLI functions that take in or return an SQL data type argument (such as SQLBindParameter(), SQLDescribeCol()).

Since LOB values can be very large, transfer of data using the piecewise sequential method provided by SQLGetData() and SQLPutData() can be quite time consuming. Applications dealing with such data will often do so in random access segments using LOB locators or via direct file input and output.

To determine if any of the LOB functions are supported for the current server, call SQLGetFunctions() with the appropriate function name argument value, or SQLGetTypeInfo() with the particular LOB data type.

Figure 8 on page 97 shows the retrieval of a character LOB (CLOB).

- The left hand side shows a locator being used to extract a character string from the CLOB, without having to transfer the entire CLOB to an application buffer. A LOB locator is fetched, which is then used as an input parameter to search the CLOB for a substring, the substring is then retrieved.
- The right hand side shows how the CLOB can be fetched directly into a file. The file is first bound to the CLOB column, and when the row is fetched, the entire CLOB value is transferred directly to a file.

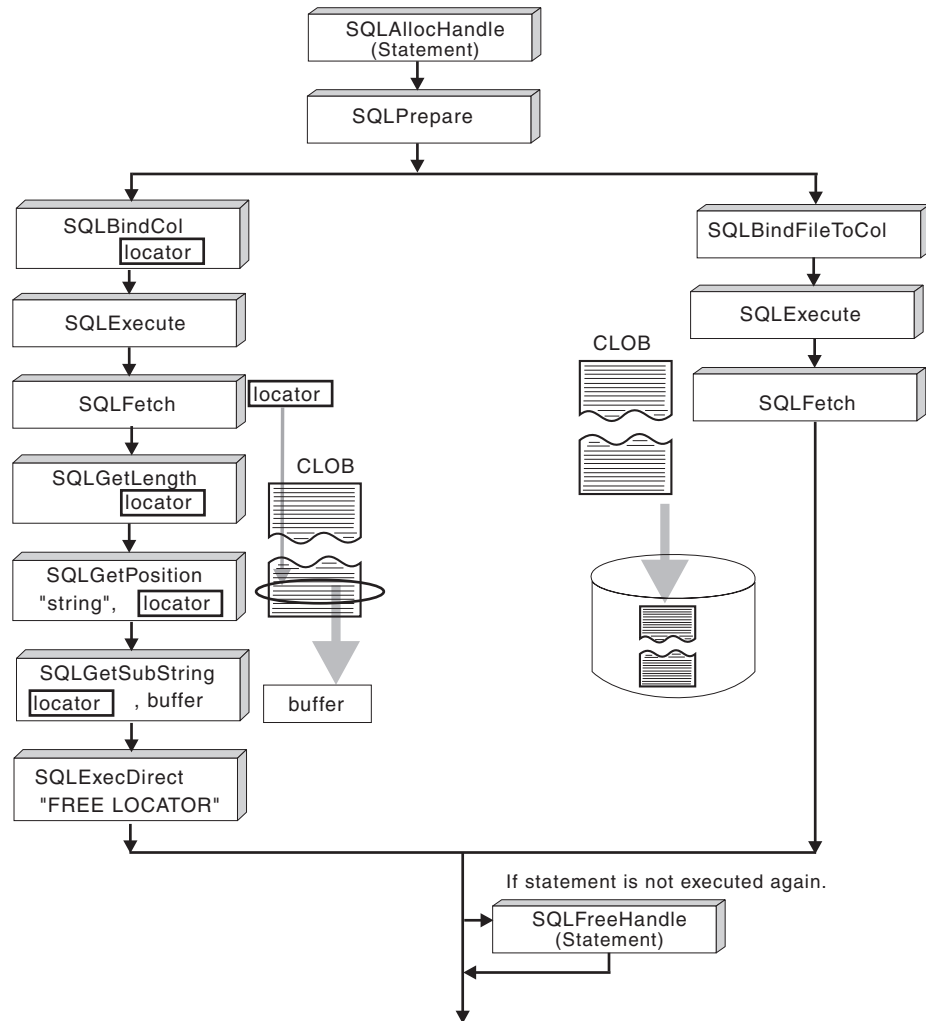


Figure 8. Fetching CLOB data

Related concepts:

- “Data retrieval in pieces in CLI applications” on page 95
- “LOB locators in CLI applications” on page 98

Related reference:

- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetFunctions function (CLI) - Get functions” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetTypeInfo function (CLI) - Get data type information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*

LOB locators in CLI applications

There are many cases where an application needs to select a large object value and operate on pieces of it, but does not need or want the entire value to be transferred from the database server into application memory. In these cases, the application can reference an individual LOB value via a large object locator (LOB locator).

A LOB locator is a token value, defined as type `SQLINTEGER`, that allows for efficient random access of a large object. When a LOB locator is used, the server performs the query and instead of placing the value of the LOB column in the result set, it updates the LOB locator with an integer that corresponds to the value of the LOB. When the application later requests the result, the application then passes the locator to the server and the server returns the LOB result.

A LOB locator is not stored in the database. It refers to a LOB value during a transaction, and does not persist beyond the transaction in which it was created. It is a simple token value created to reference a single large object *value*, and not a column in a row. There is no operation that could be performed on a locator that would have an effect on the original LOB value stored in the row.

Each of the three LOB locator types has its own C data type (`SQL_C_BLOB_LOCATOR`, `SQL_C_CLOB_LOCATOR`, `SQL_C_DBCLOB_LOCATOR`). These types are used to enable transfer of LOB locator values to and from the database server.

Locators are implicitly allocated by:

- Fetching a bound LOB column to the appropriate C locator type.
- Calling `SQLGetSubString()` and specifying that the substring be retrieved as a locator.
- Calling `SQLGetData()` on an unbound LOB column and specifying the appropriate C locator type. The C locator type must match the LOB column type or an error will occur.

LOB locators also provide an efficient method of moving data from one column of a table in a database to another column (of the same or different table) without having to pull the data first into application memory and then sending it back to the server. For example, the following `INSERT` statement inserts a LOB value that is a concatenation of 2 LOB values as represented by their locators:

```
INSERT INTO lobtable values (CAST ? AS CLOB(4k) || CAST ? AS CLOB(5k))
```

Differences between regular data types and LOB locators:

LOB locators can in general be treated as any other data type, but there are some important differences:

- Locators are generated at the server when a row is fetched and a LOB locator C data type is specified on `SQLBindCol()`, or when `SQLGetSubString()` is called to define a locator on a portion of another LOB. Only the locator is transferred to the application.
- The value of the locator is only valid within the current transaction. You cannot store a locator value and use it beyond the current transaction, even if the cursor used to fetch the LOB locator has the `WITH HOLD` attribute.
- A locator can also be freed before the end of the transaction with the `FREE LOCATOR` statement.

- Once a locator is received, the application can use `SQLGetSubString()`, to either receive a portion of the LOB value, or to generate another locator representing the sub-string. The locator value can also be used as input for a parameter marker (using `SQLBindParameter()`).

A LOB locator is not a pointer to a database position, but rather it is a reference to a LOB value: a snapshot of that LOB value. There is no association between the current position of the cursor and the row from which the LOB value was extracted. This means that even after the cursor has moved to a different row, the LOB locator (and thus the value that it represents) can still be referenced.

- `SQLGetPosition()` and `SQLGetLength()` can be used with `SQLGetSubString()` to define the sub-string.

For a given LOB column in the result set, the binding can be to a:

- storage buffer for holding the entire LOB data value,
- LOB locator, or
- LOB file reference (using `SQLBindFileToCol()`).

Related concepts:

- “Large object usage in CLI applications” on page 96
- “Parameter marker binding in CLI applications” on page 26

Related tasks:

- “Fetching LOB data with LOB locators in CLI applications” on page 99

Related reference:

- “C data types for CLI applications” on page 43
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetLength function (CLI) - Retrieve length of a string value” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetPosition function (CLI) - Return starting position of string” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetSubString function (CLI) - Retrieve portion of a string value” in *Call Level Interface Guide and Reference, Volume 2*

Fetching LOB data with LOB locators in CLI applications

The following are typical steps for fetching LOB data using a LOB locator. The examples shown in each step illustrate how using a locator to retrieve CLOB data allows a character string to be extracted from the CLOB, without having to transfer the entire CLOB to an application buffer. The LOB locator is fetched and then used as an input parameter to search the CLOB for a substring. This substring is then retrieved.

Prerequisites:

Before fetching LOB data with LOB locators, ensure you have initialized your CLI application.

Procedure:

To fetch LOB data using LOB locators:

1. Retrieve a LOB locator into an application variable using the `SQLBindCol()` or `SQLGetData()` functions. For example:

```
SQLINTEGER clobLoc ;
SQLINTEGER pcbValue ;

/* ... */
sqlrc = SQLBindCol( hstmtClobFetch, 1, SQL_C_CLOB_LOCATOR,
                  &clobLoc, 0, &pcbValue);
```

2. Fetch the locator using `SQLFetch()`:

```
sqlrc = SQLFetch( hstmtClobFetch );
```

3. Call `SQLGetLength()` to get the length of a string that is represented by a LOB locator. For example:

```
sqlrc = SQLGetLength( hstmtLocUse, SQL_C_CLOB_LOCATOR,
                    clobLoc, &clobLen, &ind );
```

4. Call `SQLGetPosition()` to get the position of a search string within a source string where the source string is represented by a LOB locator. The search string can also be represented by a LOB locator. For example:

```
sqlrc = SQLGetPosition( hstmtLocUse,
                      SQL_C_CLOB_LOCATOR,
                      clobLoc,
                      0,
                      ( SQLCHAR * ) "Interests",
                      strlen( "Interests" ),
                      1,
                      &clobPiecePos,
                      &ind );
```

5. Call `SQLGetSubString()` to retrieve the substring. For example:

```
sqlrc = SQLGetSubString( hstmtLocUse,
                       SQL_C_CLOB_LOCATOR,
                       clobLoc,
                       clobPiecePos,
                       clobLen - clobPiecePos,
                       SQL_C_CHAR,
                       buffer,
                       clobLen - clobPiecePos + 1,
                       &clobPieceLen,
                       &ind );
```

6. Free the locator. All LOB locators are implicitly freed when a transaction ends. The locator can be explicitly freed before the end of a transaction by executing the `FREE LOCATOR` statement.

Although this statement cannot be prepared dynamically, DB2 CLI will accept it as a valid statement on `SQLPrepare()` and `SQLExecDirect()`. The application uses `SQLBindParameter()` with the SQL data type argument set to the appropriate SQL and C symbolic data types. For example,

```
sqlrc = SQLSetParam( hstmtLocFree,
                   1,
                   SQL_C_CLOB_LOCATOR,
                   SQL_CLOB_LOCATOR,
                   0,
                   0,
                   &clobLoc,
                   NULL );

/* ... */
sqlrc = SQLExecDirect( hstmtLocFree, stmtLocFree, SQL_NTS );
```

Related concepts:

- “LOB locators in CLI applications” on page 98

Related tasks:

- “Initializing CLI applications” on page 18

Related reference:

- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “C data types for CLI applications” on page 43
- “FREE LOCATOR statement” in *SQL Reference, Volume 2*
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetLength function (CLI) - Retrieve length of a string value” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetPosition function (CLI) - Return starting position of string” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetSubString function (CLI) - Retrieve portion of a string value” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtlob.c -- How to read and write LOB data”

Direct file input and output for LOB handling in CLI applications

As an alternative to using LOB locators, if an application requires the entire LOB column value, it can request direct file input and output for LOBs. Database queries, updates, and inserts may involve transfer of single LOB column values into and from files. The two DB2 CLI LOB file access functions are:

SQLBindFileToCol()

Binds (associates) a LOB column in a result set with a file name.

Example:

```

SQLINTEGER    fileOption = SQL_FILE_OVERWRITE;
SQLINTEGER    fileInd = 0;
SQLSMALLINT   fileNameLength = 14;
/* ... */
SQLCHAR       fileName[14] = "";

/* ... */
rc = SQLBindFileToCol(hstmt, 1, fileName, &fileNameLength,
                      &fileOption, 14, NULL, &fileInd);

```

SQLBindFileToParam()

Binds (associates) a LOB parameter marker with a file name.

Example:

```

SQLINTEGER    fileOption = SQL_FILE_OVERWRITE;
SQLINTEGER    fileInd = 0;
SQLSMALLINT   fileNameLength = 14;
/* ... */
SQLCHAR       fileName[14] = "";

/* ... */

```

```
rc = SQLBindFileToParam(hstmt, 3, SQL_BLOB, fileName,
                        &fileNameLength, &fileOption, 14, &fileInd);
```

The file name is either the complete path name of the file (which is recommended), or a relative file name. If a relative file name is provided, it is appended to the current path (of the operating environment) of the client process. On execute or fetch, data transfer to and from the file would take place, in a similar way to that of bound application variables. A file options argument associated with these 2 functions indicates how the files are to be handled at time of transfer.

Use of `SQLBindFileToParam()` is more efficient than the sequential input of data segments using `SQLPutData()`, since `SQLPutData()` essentially puts the input segments into a temporary file and then uses the `SQLBindFileToParam()` technique to send the LOB data value to the server. Applications should take advantage of `SQLBindFileToParam()` instead of using `SQLPutData()`.

Note: DB2 CLI uses a temporary file when inserting LOB data in pieces. If the data originates in a file, the use of a temporary file can be avoided by using `SQLBindFileToParam()`. Call `SQLGetFunctions()` to query if support is provided for `SQLBindFileToParam()`, since `SQLBindFileToParam()` is not supported against servers that do not support LOBs.

Related concepts:

- “Large object usage in CLI applications” on page 96
- “LOB locators in CLI applications” on page 98

Related reference:

- “SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetFunctions function (CLI) - Get functions” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtlob.c -- How to read and write LOB data”

LOB usage in ODBC applications

Existing ODBC-compliant applications use `SQL_LONGVARCHAR` and `SQL_LONGVARBINARY` instead of the DB2 BLOB and CLOB data types. You can still access LOB columns from these ODBC-compliant applications by setting the `LongDataCompat` configuration keyword in the initialization file, or setting the `SQL_ATTR_LONGDATA_COMPAT` connection attribute using `SQLSetConnectAttr()`. Once this is done, DB2 CLI will map the ODBC long data types to the DB2 LOB data types. The `LOBMaxColumnSize` configuration keyword allows you to override the default `COLUMN_SIZE` for LOB data types.

When this mapping is in effect:

- `SQLGetTypeInfo()` will return CLOB, BLOB and DBCLOB characteristics when called with `SQL_LONGVARCHAR`, `SQL_LONGVARBINARY` or `SQL_LONGVARGRAPHIC`.
- The following functions will return `SQL_LONGVARCHAR`, `SQL_LONGVARBINARY` or `SQL_LONGVARGRAPHIC` when describing CLOB, BLOB or DBCLOB data types:
 - `SQLColumns()`
 - `SQLSpecialColumns()`
 - `SQLDescribeCol()`
 - `SQLColAttribute()`
 - `SQLProcedureColumns()`
- `LONG VARCHAR` and `LONG VARCHAR FOR BIT DATA` will continue to be described as `SQL_LONGVARCHAR` and `SQL_LONGVARBINARY`.

The default setting for `SQL_ATTR_LONGDATA_COMPAT` is `SQL_LD_COMPAT_NO`; that is, mapping is not in effect.

With mapping in effect, ODBC applications can retrieve and input LOB data by using the `SQLGetData()`, `SQLPutData()` and related functions.

Related concepts:

- “Large object usage in CLI applications” on page 96

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “LOBMaxColumnSize CLI/ODBC configuration keyword” on page 332
- “LongDataCompat CLI/ODBC configuration keyword” on page 333
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*

Bulk data manipulation

Long data for bulk inserts and updates in CLI applications

Long data can be provided for bulk inserts and updates performed by calls to `SQLBulkOperations()`.

1. When an application binds the data using `SQLBindCol()`, the application places an application-defined value, such as the column number, in the `*TargetValuePtr` buffer for data-at-execution columns. The value can be used later to identify the column.

The application places the result of the `SQL_LEN_DATA_AT_EXEC(length)` macro in the `*StrLen_or_IndPtr` buffer. If the SQL data type of the column is `SQL_LONGVARBINARY`, `SQL_LONGVARCHAR`, or a long, data source-specific data type and CLI returns “Y” for the

SQL_NEED_LONG_DATA_LEN information type in SQLGetInfo(), *length* is the number of bytes of data to be sent for the parameter; otherwise, it must be a non-negative value and is ignored.

2. When SQLBulkOperations() is called, if there are data-at-execution columns, the function returns SQL_NEED_DATA and proceeds to the next event in the sequence, described in the next item. (If there are no data-at-execution columns, the process is complete.)
3. The application calls SQLParamData() to retrieve the address of the **TargetValuePtr* buffer for the first data-at-execution column to be processed. SQLParamData() returns SQL_NEED_DATA. The application retrieves the application-defined value from the **TargetValuePtr* buffer.

Note: Although data-at-execution parameters are similar to data-at-execution columns, the value returned by SQLParamData() is different for each.

Data-at-execution columns are columns in a rowset for which data will be sent with SQLPutData() when a row is updated or inserted with SQLBulkOperations(). They are bound with SQLBindCol(). The value returned by SQLParamData() is the address of the row in the **TargetValuePtr* buffer that is being processed.

4. The application calls SQLPutData() one or more times to send data for the column. More than one call is needed if all the data value cannot be returned in the **TargetValuePtr* buffer specified in SQLPutData(); note that multiple calls to SQLPutData() for the same column are allowed only when sending character C data to a column with a character, binary, or data source-specific data type or when sending binary C data to a column with a character, binary, or data source-specific data type.
5. The application calls SQLParamData() again to signal that all data has been sent for the column.
 - If there are more data-at-execution columns, SQLParamData() returns SQL_NEED_DATA and the address of the *TargetValuePtr* buffer for the next data-at-execution column to be processed. The application repeats steps 4 and 5 above.
 - If there are no more data-at-execution columns, the process is complete. If the statement was executed successfully, SQLParamData() returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO; if the execution failed, it returns SQL_ERROR. At this point, SQLParamData() can return any SQLSTATE that can be returned by SQLBulkOperations().

If the operation is canceled, or an error occurs in SQLParamData() or SQLPutData(), after SQLBulkOperations() returns SQL_NEED_DATA, and before data is sent for all data-at-execution columns, the application can call only SQLCancel(), SQLGetDiagField(), SQLGetDiagRec(), SQLGetFunctions(), SQLParamData(), or SQLPutData() for the statement or the connection associated with the statement. If it calls any other function for the statement or the connection associated with the statement, the function returns SQL_ERROR and SQLSTATE HY010 (Function sequence error).

If the application calls SQLCancel() while CLI still needs data for data-at-execution columns, CLI cancels the operation. The application can then call SQLBulkOperations() again; canceling does not affect the cursor state or the current cursor position.

Related reference:

- “CLI function return codes” on page 50
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetInfo function (CLI) - Get general information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLParamData function (CLI) - Get next parameter for which a data value is needed” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*

Retrieving bulk data with bookmarks using SQLBulkOperations() in CLI applications

You can retrieve, or fetch, bulk data using bookmarks and the DB2 CLI `SQLBulkOperations()` function.

Prerequisites:

Before fetching bulk data using bookmarks and `SQLBulkOperations()`, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating with bookmarks.

Procedure:

To perform bulk fetches using bookmarks with `SQLBulkOperations()`:

1. Set the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE` using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.
3. Set the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows you want to fetch by calling `SQLSetStmtAttr()`.
4. Call `SQLBindCol()` to bind the data you want to fetch.
The data is bound to an array with a size equal to the value of `SQL_ATTR_ROW_ARRAY_SIZE`.
5. Call `SQLBindCol()` to bind column 0, the bookmark column.
6. Copy the bookmarks for rows you want to fetch into the array bound to column 0.

Note: The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE`, or the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should be a null pointer.

7. Fetch the data by calling `SQLBulkOperations()` with an *Operation* argument of `SQL_FETCH_BY_BOOKMARK`.

If the application has set the `SQL_ATTR_ROW_STATUS_PTR` statement attribute, then it can inspect this array to see the result of the operation.

Related concepts:

- “Bookmarks in CLI applications” on page 78

Related tasks:

- “Deleting bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 109
- “Initializing CLI applications” on page 18
- “Inserting bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 106
- “Updating bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 107

Related reference:

- “`SQLBindCol` function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “`SQLBulkOperations` function (CLI) - Add, update, delete or fetch a set of rows” in *Call Level Interface Guide and Reference, Volume 2*
- “`SQLSetStmtAttr` function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Inserting bulk data with bookmarks using `SQLBulkOperations()` in CLI applications

You can insert data in bulk with bookmarks using `SQLBulkOperations()`.

Prerequisites:

Before inserting bulk data with `SQLBulkOperations()`, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating with bookmarks.

Procedure:

To perform a bulk data insert using `SQLBulkOperations()`:

1. Set the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE` using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.
3. Set the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows you want to insert using `SQLSetStmtAttr()`.
4. Call `SQLBindCol()` to bind the data you want to insert.

The data is bound to an array with a size equal to the value of `SQL_ATTR_ROW_ARRAY_SIZE`, set in the previous step.

Note: The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ATTR_ROW_STATUS_PTR` should be a null pointer.

5. Insert the data by calling `SQLBulkOperations()` with `SQL_ADD` as the *Operation* argument.

CLI will update the bound column 0 buffers with the bookmark values for the newly inserted rows. For this to occur, the application must have set the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE` before executing the statement.

Note: If `SQLBulkOperations()` is called with an *Operation* argument of `SQL_ADD` on a cursor that contains duplicate columns, an error is returned.

Related concepts:

- “Bookmarks in CLI applications” on page 78

Related tasks:

- “Deleting bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 109
- “Initializing CLI applications” on page 18
- “Retrieving bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 105
- “Updating bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 107

Related reference:

- “`SQLBindCol` function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “`SQLBulkOperations` function (CLI) - Add, update, delete or fetch a set of rows” in *Call Level Interface Guide and Reference, Volume 2*
- “`SQLSetStmtAttr` function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Updating bulk data with bookmarks using `SQLBulkOperations()` in CLI applications

You can update data in bulk with bookmarks using `SQLBulkOperations()`.

Prerequisites:

Before updating data in bulk, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating with bookmarks.

Procedure:

To update data in bulk:

1. Set the `SQL_ATTR_USE_BOOKMARKS` statement attribute to `SQL_UB_VARIABLE` using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.
3. Set the `SQL_ATTR_ROW_ARRAY_SIZE` statement attribute to the number of rows you want to update using `SQLSetStmtAttr()`.
4. Call `SQLBindCol()` to bind the data you want to update.
The data is bound to an array with a size equal to the value of `SQL_ATTR_ROW_ARRAY_SIZE`, set in the previous step.
5. Bind the bookmark column to column 0 by calling `SQLBindCol()`.
6. Copy the bookmarks for rows that you want to update into the array bound to column 0.
7. Update the data in the bound buffers.

Note: The size of the array pointed to by the `SQL_ATTR_ROW_STATUS_PTR` statement attribute should either be equal to `SQL_ATTR_ROW_ARRAY_SIZE` or `SQL_ATTR_ROW_STATUS_PTR` should be a null pointer.

8. Update the data by calling `SQLBulkOperations()` with an *Operation* argument of `SQL_UPDATE_BY_BOOKMARK`.

Note: If the application has set the `SQL_ATTR_ROW_STATUS_PTR` statement attribute, then it can inspect this array to see the result of the operation.

9. Optional: Verify that the update has occurred by calling `SQLBulkOperations()` with an *Operation* argument of `SQL_FETCH_BY_BOOKMARK`. This will fetch the data into the bound application buffers.

If data has been updated, CLI changes the value in the row status array for the appropriate rows to `SQL_ROW_UPDATED`.

Note: If `SQLBulkOperations()` is called with an *Operation* argument of `SQL_UPDATE_BY_BOOKMARK` on a cursor that contains duplicate columns, an error is returned.

Related concepts:

- “Bookmarks in CLI applications” on page 78

Related tasks:

- “Deleting bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 109
- “Initializing CLI applications” on page 18
- “Inserting bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 106
- “Retrieving bulk data with bookmarks using `SQLBulkOperations()` in CLI applications” on page 105

Related reference:

- “`SQLBindCol` function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Deleting bulk data with bookmarks using SQLBulkOperations() in CLI applications

You can use SQLBulkOperations() and bookmarks to delete data in bulk.

Prerequisites:

Before deleting data in bulk, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call SQLFetch() or SQLFetchScroll() to retrieve the bookmarks before updating by bookmarks.

Procedure:

To perform bulk deletions using bookmarks and SQLBulkOperations():

1. Set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE using SQLSetStmtAttr().
2. Execute a query that returns a result set.
3. Set the SQL_ATTR_ROW_ARRAY_SIZE statement attribute to the number of rows you want to delete.
4. Bind the bookmark column to column 0 by calling SQLBindCol().
5. Copy the bookmarks for the rows you want to delete into the array bound to column 0.

Note: The size of the array pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute should either be equal to SQL_ATTR_ROW_ARRAY_SIZE, or the SQL_ATTR_ROW_STATUS_PTR statement attribute should be a null pointer.

6. Perform the deletion by calling SQLBulkOperations() with an *Operation* argument of SQL_DELETE_BY_BOOKMARK.

If the application has set the SQL_ATTR_ROW_STATUS_PTR statement attribute, then it can inspect this array to see the result of the operation.

Related concepts:

- “Bookmarks in CLI applications” on page 78

Related tasks:

- “Initializing CLI applications” on page 18
- “Inserting bulk data with bookmarks using SQLBulkOperations() in CLI applications” on page 106

- “Retrieving bulk data with bookmarks using SQLBulkOperations() in CLI applications” on page 105
- “Updating bulk data with bookmarks using SQLBulkOperations() in CLI applications” on page 107

Related reference:

- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Importing data with the CLI LOAD utility in CLI applications

The CLI LOAD functionality provides an interface to the IBM DB2 LOAD utility from CLI. This functionality allows you to insert data in CLI using LOAD instead of array insert. This option can yield significant performance benefits when large amounts of data need to be inserted. Because this interface invokes LOAD, the same consideration given for using LOAD should also be taken into account when using the CLI LOAD interface.

Prerequisites:

Before importing data with the CLI LOAD utility, ensure you have initialized your CLI application.

Restrictions:

- Unlike the IBM DB2 LOAD utility, the CLI LOAD utility does not load data directly from an input file. Instead, if desired, the application should retrieve the data from the input file and insert it into the appropriate application parameters that correspond to the parameter markers in the prepared statement.
- If the prepared SQL statement for inserting data contains a SELECT clause, parameter markers are not supported.
- The prepared SQL statement for inserting data must include parameter markers for all columns in the target table, unless a fullselect is used instead of the VALUES clause in the INSERT statement.
- The insertion of data is non-atomic because the load utility precludes atomicity. LOAD might not be able to successfully insert all the rows passed to it. For example, if a unique key constraint is violated by a row being inserted, LOAD will not insert this row but will continue loading the remaining rows.
- A COMMIT will be issued by LOAD. Therefore, if the insertion of the data completes successfully, the LOAD and any other statements within the transaction cannot be rolled back.
- The error reporting for the CLI LOAD interface differs from that of array insert. Non-severe errors or warnings, such as errors with specific rows, will only appear in the LOAD message file.

Procedure:

To import data using the CLI LOAD utility:

1. Specify the statement attribute `SQL_ATTR_USE_LOAD_API` in `SQLSetStmtAttr()` with one of the following supported values:

SQL_USE_LOAD_INSERT

Use the LOAD utility to append to existing data in the table.

SQL_USE_LOAD_REPLACE

Use the LOAD utility to replace existing data in the table.

For example, the following call indicates that the CLI LOAD utility will be used to add to the existing data in the table:

```
SQLSetStmtAttr (hStmt, SQL_ATTR_USE_LOAD_API,  
                (SQLPOINTER) SQL_USE_LOAD_INSERT, 0);
```

Note: When `SQL_USE_LOAD_INSERT` or `SQL_USE_LOAD_REPLACE` is set, no other CLI functions except for the following can be called until `SQL_USE_LOAD_OFF` is set (see Step 3 below):

- `SQLBindParameter()`
 - `SQLExecute()`
 - `SQLExtendedBind()`
 - `SQLParamOptions()`
 - `SQLSetStmtAttr()`
2. Create a structure of type `db2LoadStruct` and specify the desired load options through this structure. Set the `SQL_ATTR_LOAD_INFO` statement attribute to a pointer to this structure.

3. Issue `SQLExecute()` on the prepared SQL statement for the data to be inserted. The INSERT SQL statement can be a fullselect which allows data to be loaded from a table using the SELECT statement. With a single execution of the INSERT statement, all of the data from the SELECT is loaded. The following example shows how a fullselect statement loads data from one table into another:

```
SQLPrepare (hStmt,  
            (SQLCHAR *) "INSERT INTO tableB SELECT * FROM tableA",  
            SQL_NTS);  
SQLExecute (hStmt);
```

4. Call `SQLSetStmtAttr()` with `SQL_USE_LOAD_OFF`. This ends the processing of data using the LOAD utility. Subsequently, regular CLI array insert will be in effect until `SQL_ATTR_USE_LOAD_API` is set again (see Step 1).
5. Optional: Query the results of the completed CLI LOAD operation by calling `SQLGetStmtAttr()` with any of the following statement attributes:
 - `SQL_ATTR_LOAD_ROWS_COMMITTED_PTR`: A pointer to an integer that represents the total number of rows processed. This value equals the number of rows successfully loaded and committed to the database, plus the number of skipped and rejected rows.
 - `SQL_ATTR_LOAD_ROWS_DELETED_PTR`: A pointer to an integer that represents the number of duplicate rows deleted.
 - `SQL_ATTR_LOAD_ROWS_LOADED_PTR`: A pointer to an integer that represents the number of rows loaded into the target table.
 - `SQL_ATTR_LOAD_ROWS_READ_PTR`: A pointer to an integer that represents the number of rows read.
 - `SQL_ATTR_LOAD_ROWS_REJECTED_PTR`: A pointer to an integer that represents the number of rows that could not be loaded.

- `SQL_ATTR_LOAD_ROWS_SKIPPED_PTR`: A pointer to an integer that represents the number of rows skipped before the CLI LOAD operation began.

Related tasks:

- “Binding parameter markers in CLI applications with column-wise array input” on page 81
- “Binding parameter markers in CLI applications with row-wise array input” on page 82
- “Initializing CLI applications” on page 18

Related reference:

- “db2Load API - Load data into a table” in *Administrative API Reference*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “LOAD command” in *Command Reference*

Related samples:

- “tblload.c -- How to insert data using the CLI LOAD utility ”

Chapter 8. Stored procedures

Calling stored procedures from CLI applications

CLI applications invoke stored procedures by executing the CALL procedure SQL statement. This topic describes how to call stored procedures from CLI applications.

Prerequisites:

Before calling a stored procedure, ensure that you have initialized your CLI application.

Restrictions:

If the stored procedure being called is uncataloged, ensure that it does not call any of the CLI schema functions. Calling CLI schema functions from uncataloged stored procedures is not supported.

The CLI schema functions are: `SQLColumns()`, `SQLColumnPrivileges()`, `SQLForeignKeys()`, `SQLPrimaryKeys()`, `SQLProcedureColumns()`, `SQLProcedures()`, `SQLSpecialColumns()`, `SQLStatistics()`, `SQLTables()`, and `SQLTablePrivileges()`.

Procedure:

To call a stored procedure:

1. Declare application host variables corresponding to each of the IN, INOUT, and OUT parameters of the stored procedure. Ensure the application variable data types and lengths match the data types and lengths of the arguments in the stored procedure signature. DB2 CLI supports calling stored procedures with all SQL types as parameter markers.
2. Initialize the IN, INOUT, and OUT parameter application variables.
3. Issue the CALL SQL statement. For example:

```
SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (?)";
```

For optimal performance, applications should use parameter markers for stored procedure arguments in the CALL procedure string and then bind the host variables to those parameter markers. If inbound stored procedure arguments must be specified as string literals rather than parameter markers, however, include the ODBC call escape clause delimiters { } in the CALL procedure statement. For example:

```
SQLCHAR *stmt = (SQLCHAR *)"{CALL IN_PARAM (123, 'Hello World!')}";
```

When string literals and the ODBC escape clause are used in a CALL procedure statement, the string literals can only be specified as IN mode stored procedure arguments. INOUT and OUT mode stored procedure arguments must still be specified using parameter markers.

4. Optional: Prepare the CALL statement by calling `SQLPrepare()`.
5. Bind each parameter of the CALL procedure statement by calling `SQLBindParameter()`.

Note: Ensure each parameter is bound correctly (to SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT), otherwise unexpected results could occur when the CALL procedure statement is executed. This would happen, for example, if an input parameter was incorrectly bound with an *InputOutputType* of SQL_PARAM_OUTPUT.

6. Execute the CALL procedure statement using SQLExecDirect(), or if the CALL procedure statement was prepared in step 4, SQLExecute().

Note: If an application or thread that has invoked a stored procedure is terminated before the stored procedure completes, execution of the stored procedure will also be terminated. It is important that a stored procedure contain logic to ensure that the database is in both a consistent and desirable state if the stored procedure is terminated prematurely.

7. Check the return code of SQLExecDirect() or SQLExecute() when the function has returned to determine if any errors occurred during execution of either the CALL procedure statement or the stored procedure. If the return code is SQL_SUCCESS_WITH_INFO or SQL_ERROR, use the CLI diagnostic functions SQLGetDiagRec() and SQLGetDiagField() to determine why the error occurred. If a stored procedure has executed successfully, any variables bound as OUT parameters may contain data that the stored procedure has passed back to the CLI application. If applicable, the stored procedure may also return one or more result sets through non-scrollable cursors. CLI applications should process stored procedure result sets as they would process result sets generated by executing SELECT statements.

Note: If a CLI application is unsure of the number or type of parameters in a result set returned by a stored procedure, the SQLNumResultCols(), SQLDescribeCol(), and SQLColAttribute() functions can be called (in this order) on the result set to determine this information.

Once you have executed the CALL statement, you can retrieve result sets from the stored procedure if applicable.

Note:

The numeric month and day parts of a DATETIME data type value will appear to be reversed in procedure result sets that are returned to DB2 CLI applications if the values are not returned in ISO format. For example, this can happen if a local format is used instead. To ensure that DATETIME data type value information is correctly interpreted by a client application, the procedures should be bound to the database with a locale-independent DATETIME format such as ISO. For example:

```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
```

Note:

DB2 CLI packages are automatically bound to databases when the databases are created or migrated. If a FixPak is applied to either the client or the server, however, then you must rebind db2cli.lst by issuing the following command:

UNIX

```
db2 bind <BNDPATH>/@db2cli.lst blocking all grant public
```

Windows

```
db2 bind "%DB2PATH%\bnd\@db2cli.lst" blocking all grant public
```

Related concepts:

- “Routines: Procedures” in *SQL Guide*

Related tasks:

- “Binding parameter markers in CLI applications” on page 29
- “Initializing CLI applications” on page 18
- “Preparing and executing SQL statements in CLI applications” on page 24
- “Setting up the CLI environment” on page 209

Related reference:

- “CALL statement” in *SQL Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI function return codes” on page 50
- “DB2 CLI bind files and package names” on page 203
- “DB2 CLI stored procedure commit behavior” on page 115

Related samples:

- “spcall.c -- Call individual stored procedures”
- “spclient.c -- Call various stored procedures”
- “spclires.c -- Contrast stored procedure multiple result set handling methods”
- “spserver.c -- Definition of various types of stored procedures”

DB2 CLI stored procedure commit behavior

The commit behavior of SQL statements, both in a DB2 CLI client application and in the called stored procedure running on a DB2 server, depends on the commit combinations applied in the application and the stored procedure. The possible combinations and the resulting commit behavior are described in the following table.

Table 10. DB2 CLI Stored procedure commit behavior

CLI client	Stored procedure	Commit behavior
autocommit on	autocommit on	All successfully executed SQL statements in the stored procedure are committed, even if other SQL statements in the stored procedure fail and an error or warning SQLCODE is returned to the CALL statement.

Table 10. DB2 CLI Stored procedure commit behavior (continued)

CLI client	Stored procedure	Commit behavior
autocommit on	autocommit off	If the stored procedure returns an SQLCODE ≥ 0 , all successfully executed SQL statements in the stored procedure are committed. Otherwise, all SQL statements in the stored procedure are rolled back.
autocommit on	manual commit	All successfully executed SQL statements in the stored procedure that are manually committed will not be rolled back, even if an error SQLCODE is returned to the CALL statement. Note: If the stored procedure returns an SQLCODE ≥ 0 , any successfully executed SQL statements in the stored procedure that occur after the last manual commit will be committed; otherwise, they will be rolled back to the manual commit point.
autocommit off	autocommit on	All successfully executed SQL statements in the stored procedure are committed and will not be rolled back, even if an error SQLCODE is returned to the CALL statement. In addition, all uncommitted and successfully executed SQL statements in the CLI client application up to and including the CALL statement are committed. Note: Exercise caution when using this commit combination in a multi-SQL statement client-side transaction, because the transaction cannot be fully rolled back after the CALL statement has been issued.
autocommit off	autocommit off	If the stored procedure returns an SQLCODE ≥ 0 , all successfully executed SQL statements in the stored procedure will be committed when the transaction that includes the CALL statement is committed. Otherwise, all SQL statements in the stored procedure will be rolled back when the transaction that includes the CALL statement is rolled back.
autocommit off	manual commit	All successfully executed SQL statements in the stored procedure that are manually committed will not be rolled back, even if an error SQLCODE is returned to the CALL statement. In addition, all uncommitted and successfully executed SQL statements in the CLI client application up to the CALL statement are committed. Note: If the stored procedure returns an SQLCODE ≥ 0 , any successfully executed SQL statements within the stored procedure that occur after the last manual commit will be committed; otherwise, they will be rolled back to the manual commit point. Note: Exercise caution when using this commit combination in a multi-SQL statement client-side transaction, because the transaction cannot be fully rolled back after the CALL statement has been issued.

Related concepts:

- “Routines: Procedures” in *SQL Guide*

Related tasks:

- “Calling stored procedures from CLI applications” on page 113

Related reference:

- “CALL statement” in *SQL Reference, Volume 2*

Chapter 9. XML data

XML data handling in CLI applications - Overview

DB2 CLI applications can retrieve and store XML data using the SQL_XML data type. This data type corresponds to the native XML data type of the DB2 database, which is used to define columns that store well-formed XML documents. The SQL_XML type can be bound to the following C types: SQL_C_BINARY, SQL_C_CHAR, SQL_C_WCHAR, and SQL_C_DBCHAR. Using the default SQL_C_BINARY type, however, instead of character types, is recommended to avoid possible data loss or corruption resulting from code page conversion when character types are used.

To store XML data in an XML column, bind a binary (SQL_C_BINARY) or character (SQL_C_CHAR, SQL_C_WCHAR, or SQL_C_DBCHAR) buffer that contains the XML value to the SQL_XML SQL type and execute the INSERT or UPDATE SQL statements. To retrieve XML data from the database, bind the result set to a binary (SQL_C_BINARY) or character (SQL_C_CHAR, SQL_C_WCHAR, or SQL_C_DBCHAR) type. Character types should be used with caution because of encoding issues.

When an XML value is retrieved into an application data buffer, the DB2 server performs an implicit serialization on the XML value to convert it from its stored hierarchical form to the serialized string form. For character typed buffers, the XML value is implicitly serialized to the application character code page associated with the character type.

By default, an XML declaration is included in the output serialized string. This default behavior can be changed by setting the SQL_ATTR_XML_DECLARATION statement or connection attribute, or by setting the XMLDeclaration CLI/ODBC configuration keyword in the db2cli.ini file.

XQuery expressions and SQL/XML functions can be issued and executed in DB2 CLI applications. SQL/XML functions are issued and executed like any other SQL statements. XQuery expressions must either be prepended with the case-insensitive keyword "XQUERY", or the SQL_ATTR_XQUERY_STATEMENT statement attribute must be set for the statement handle associated with the XQuery expression.

Related concepts:

- "Application programming language support for XML" in *XML Guide*
- "XML column inserts and updates in CLI applications" on page 118
- "XML data encoding" in *XML Guide*
- "XML data retrieval in CLI applications" on page 119
- "XML data type" in *XML Guide*

Related reference:

- "Connection attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*
- "Statement attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*

- “XMLDeclaration CLI/ODBC configuration keyword” on page 392

XML column inserts and updates in CLI applications

When you update or insert data into XML columns of a table, the input data must be in the serialized string format.

For XML data, when you use `SQLBindParameter()` to bind parameter markers to input data buffers, you can specify the data type of the input data buffer as `SQL_C_BINARY`, `SQL_C_CHAR`, `SQL_C_DBCHAR` or `SQL_C_WCHAR`.

When you bind a data buffer that contains XML data as `SQL_C_BINARY`, DB2 CLI processes the XML data as internally encoded data. This is the preferred method because it avoids the overhead and potential data loss of character conversion when character types are used.

Important: If the XML data is encoded in an encoding scheme and CCSID other than the application code page encoding scheme, you need to include internal encoding in the data and bind the data as `SQL_C_BINARY` to avoid character conversion.

When you bind a data buffer that contains XML data as `SQL_C_CHAR`, `SQL_C_DBCHAR` or `SQL_C_WCHAR`, DB2 CLI processes the XML data as externally encoded data. DB2 CLI determines the encoding of the data as follows:

- If the C type is `SQL_C_WCHAR`, DB2 CLI assumes that the data is encoded as UCS-2.
- If the C type is `SQL_C_CHAR` or `SQL_C_DBCHAR`, DB2 CLI assumes that the data is encoded in the application code page encoding scheme.

If you want the database server to implicitly parse the data before storing it in an XML column, the parameter marker data type in `SQLBindParameter()` should be specified as `SQL_XML`.

Implicit parsing is recommended, because explicit parsing of a character type with `XMLPARSE` can introduce encoding issues.

The following example shows how to update XML data in an XML column using the recommended `SQL_C_BINARY` type.

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);
```

Related concepts:

- “Application programming language support for XML” in *XML Guide*
- “Creation of tables with XML columns” in *XML Guide*
- “Updates of XML columns” in *XML Guide*

- “XML data encoding” in *XML Guide*
- “XML data type” in *XML Guide*
- “XML parsing” in *XML Guide*
- “XML data handling in CLI applications - Overview” on page 117

Related reference:

- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “XMLPARSE scalar function” in *SQL Reference, Volume 1*
- “Data conversions supported in CLI” on page 397

XML data retrieval in CLI applications

When you select data from XML columns in a table, the output data is in the serialized string format.

For XML data, when you use `SQLBindCol()` to bind columns in a query result set to application variables, you can specify the data type of the application variables as `SQL_C_BINARY`, `SQL_C_CHAR`, `SQL_C_DBCHAR` or `SQL_C_WCHAR`. When retrieving a result set from an XML column, it is recommended that you bind your application variable to the `SQL_C_BINARY` type. Binding to character types can result in possible data loss resulting from code page conversion. Data loss can occur when characters in the source code page cannot be represented in the target code page. Binding your variable to the `SQL_C_BINARY` C type avoids these issues.

XML data is returned to the application as internally encoded data. DB2 CLI determines the encoding of the data as follows:

- If the C type is `SQL_C_BINARY`, DB2 CLI returns the data in the UTF-8 encoding scheme.
- If the C type is `SQL_C_CHAR` or `SQL_C_DBCHAR`, DB2 CLI returns the data in the application code page encoding scheme.
- If the C type is `SQL_C_WCHAR`, DB2 CLI returns the data in the UCS-2 encoding scheme.

The database server performs an implicit serialization of the data before returning it to the application. You can explicitly serialize the XML data to a specific data type by calling the `XMLSERIALIZE` function. Implicit serialization is recommended, however, because explicitly serializing to character types with `XMLSERIALIZE` can introduce encoding issues.

The following example shows how to retrieve XML data from an XML column into a binary application variable.

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)
```

```

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8

```

Related concepts:

- “XML data handling in CLI applications - Overview” on page 117
- “Application programming language support for XML” in *XML Guide*
- “XML data encoding” in *XML Guide*
- “XML data querying” in *XML Guide*
- “XML data type” in *XML Guide*
- “XML serialization” in *XML Guide*

Related reference:

- “Data conversions supported in CLI” on page 397
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLCloseCursor function (CLI) - Close cursor and discard pending results” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetch function (CLI) - Fetch next row” in *Call Level Interface Guide and Reference, Volume 2*
- “XMLSERIALIZE scalar function” in *SQL Reference, Volume 1*

Changing of default XML type handling in CLI applications

DB2 CLI supports CLI/ODBC configuration keywords that provide compatibility for applications that do not expect the default types returned when describing or specifying SQL_C_DEFAULT for XML columns and parameter markers. Older CLI and ODBC applications might not recognize or expect the default SQL_XML type when describing XML columns or parameters. Some CLI or ODBC applications might also expect a default type other than SQL_C_BINARY for XML columns and parameter markers. To provide compatibility for these types of applications, DB2 CLI supports the MapXMLDescribe and MapXMLCDefault keywords.

MapXMLDescribe specifies which SQL data type is returned when XML columns or parameter markers are described.

MapXMLCDefault specifies the C type that is used when SQL_C_DEFAULT is specified for XML columns and parameter markers in DB2 CLI functions.

Related concepts:

- “Data types and data conversion in CLI applications” on page 39
- “XML data handling in CLI applications - Overview” on page 117
- “XML data type” in *XML Guide*

Related reference:

- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*
- “MapXMLCDefault CLI/ODBC configuration keyword” on page 343
- “MapXMLDescribe CLI/ODBC configuration keyword” on page 344

Chapter 10. Compound SQL

Executing compound SQL statements in CLI applications 123

Return codes for compound SQL in CLI applications 125

Executing compound SQL statements in CLI applications

Compound SQL allows multiple SQL statements to be grouped into a single executable block. This block of statements, together with any input parameter values, can then be executed in a single continuous stream, reducing the execution time and network traffic.

Restrictions:

- Compound SQL does not guarantee the order in which the substatements are executed, therefore there must not be any dependencies among the substatements.
- Compound SQL statements cannot be nested.
- The BEGIN COMPOUND and END COMPOUND statements must be executed with the same statement handle.
- The value specified in the STOP AFTER FIRST ? STATEMENTS clause of the BEGIN COMPOUND SQL statement must be of type SQL_INTEGER, and you can only bind an application buffer of type SQL_C_INTEGER or SQL_C_SMALLINT for this value.
- Each substatement must have its own statement handle.
- All statement handles must belong to the same connection and have the same isolation level.
- Atomic array input is not supported within a BEGIN COMPOUND and END COMPOUND block of SQL statements. Atomic array input refers to the behavior where all inserts will be undone if any single insert fails.
- All statement handles must remain allocated until the END COMPOUND statement is executed.
- SQLEndTran() cannot be called for the same connection or any connect requests between BEGIN COMPOUND and END COMPOUND.
- Only the following functions may be called using the statement handles allocated for the compound substatements:
 - SQLAllocHandle()
 - SQLBindParameter()
 - SQLBindFileToParam()
 - SQLExecute()
 - SQLParamData()
 - SQLPrepare()
 - SQLPutData()

Procedure:

To execute compound SQL statements in CLI applications:

1. Allocate a parent statement handle. For example:
`SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtparent);`

2. Allocate statement handles for each of the compound substatements. For example:


```
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub1);
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub2);
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub3);
```
3. Prepare the substatements. For example:


```
SQLPrepare (hstmtsub1, stmt1, SQL_NTS);
SQLPrepare (hstmtsub2, stmt2, SQL_NTS);
SQLPrepare (hstmtsub3, stmt3, SQL_NTS);
```
4. Execute the BEGIN COMPOUND statement using the parent statement handle. For example:


```
SQLExecDirect (hstmtparent, (SQLCHAR *) "BEGIN COMPOUND NOT ATOMIC STATIC",
SQL_NTS);
```
5. If this is an atomic compound SQL operation, execute the substatements using the SQLExecute() function only. For example:


```
SQLExecute (hstmtsub1);
SQLExecute (hstmtsub2);
SQLExecute (hstmtsub3);
```

Note: All statements to be executed inside an atomic compound block must first be prepared. Attempts to use the SQLExecDirect() function within an atomic compound block will result in errors.
6. Execute the END COMPOUND statement using the parent statement handle. For example:


```
SQLExecDirect (hstmtparent, (SQLCHAR *) "END COMPOUND NOT ATOMIC STATIC",
SQL_NTS);
```
7. Optional: If you used an input parameter value array, call SQLRowCount() with the parent statement handle to retrieve the aggregate number of rows affected by all elements of the input array. For example:


```
SQLRowCount (hstmtparent, &numRows);
```
8. Free the handles of the substatements. For example:


```
SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub1);
SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub2);
SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub3);
```
9. Free the parent statement handle when you have finished using it. For example:


```
SQLFreeHandle (SQL_HANDLE_STMT, hstmtparent);
```

If the application is not operating in auto-commit mode and the COMMIT option is not specified, the sub-statements will not be committed. If the application is operating in auto-commit mode, however, then the sub-statements will be committed at END COMPOUND, even if the COMMIT option is not specified.

Related tasks:

- “Allocating statement handles in CLI applications” on page 22
- “Binding parameter markers in CLI applications with column-wise array input” on page 81
- “Binding parameter markers in CLI applications with row-wise array input” on page 82
- “Freeing statement resources in CLI applications” on page 37
- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “Compound SQL (Dynamic) statement” in *SQL Reference, Volume 2*
- “COMMIT statement” in *SQL Reference, Volume 2*
- “ROLLBACK statement” in *SQL Reference, Volume 2*
- “SQLAllocHandle function (CLI) - Allocate handle” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLRowCount function (CLI) - Get row count” in *Call Level Interface Guide and Reference, Volume 2*
- “Return codes for compound SQL in CLI applications” on page 125

Related samples:

- “dbuse.c -- How to use a database”

Return codes for compound SQL in CLI applications

Return codes are generated on the call to `SQLExecute()` or `SQLExecDirect()` for the `END COMPOUND` statement. The following lists the return codes for `ATOMIC` and `NOT ATOMIC` compound statements:

ATOMIC

- `SQL_SUCCESS`: all substatements have executed without any warnings or errors.
- `SQL_SUCCESS_WITH_INFO`: all substatements executed successfully with one or more warnings. Call `SQLGetDiagRec()` or `SQLGetDiagField()` to retrieve additional information on the error or warning. The handle used by `SQLGetDiagRec()` or `SQLGetDiagField()` must be the same one used to process the `BEGIN COMPOUND` and `END COMPOUND` statements.
- `SQL_NO_DATA_FOUND`: `BEGIN COMPOUND` and `END COMPOUND` statements executed without any substatements, or none of the substatements affected any rows.
- `SQL_ERROR`: one or more substatements failed and all substatements were rolled back.

NOT ATOMIC

- `SQL_SUCCESS`: all substatements executed without any errors.
- `SQL_SUCCESS_WITH_INFO`: the `COMPOUND` statement executed with one or more warnings returned by one or more substatements. Call `SQLGetDiagRec()` or `SQLGetDiagField()` to retrieve additional information on the error or warning. The handle used by `SQLGetDiagRec()` or `SQLGetDiagField()` must be the same one used to process the `BEGIN COMPOUND` and `END COMPOUND` statements.
- `SQL_NO_DATA_FOUND`: the `BEGIN COMPOUND` and `END COMPOUND` statements executed without any substatements, or none of the substatements affected any rows.

- `SQL_ERROR`: the `COMPOUND` statement failed. At least one substatement returned an error. Examine the `SQLCA` to determine which statements failed.

Related tasks:

- “Executing compound SQL statements in CLI applications” on page 123

Related reference:

- “Compound SQL (Dynamic) statement” in *SQL Reference, Volume 2*
- “SQLCA (SQL communications area)” in *SQL Reference, Volume 1*
- “SQLError function (CLI) - Retrieve error information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetSQLCA function (CLI) - Get SQLCA data structure” in *Call Level Interface Guide and Reference, Volume 2*

Chapter 11. Asynchronous execution of CLI functions

Asynchronous execution of CLI functions	127
Executing functions asynchronously in CLI applications	129

Asynchronous execution of CLI functions

DB2 CLI can run a subset of CLI functions asynchronously. For these functions, the DB2 CLI driver returns control to the application after calling the function but before that function has finished executing.

Asynchronous execution is possible for those functions that normally send a request to the server and then wait for a response. The functions return `SQL_STILL_EXECUTING` each time they are called until they are finished running, at which point they return a different value (for example, `SQL_SUCCESS`). Rather than waiting for a response, a function executing asynchronously returns control to the application. The application can then perform other tasks and poll the function until a return code other than `SQL_STILL_EXECUTING` is returned. Refer to the `SQL_ATTR_ASYNC_ENABLE` connection or statement attribute for a list of functions that can be executed asynchronously.

In order for an application to run CLI functions asynchronously, the application must include the following:

1. A call to the function `SQLGetInfo()` with the `SQL_ASYNC_MODE` option to ensure support for asynchronous calls.
2. A call to `SQLSetConnectAttr()` or `SQLSetStmtAttr()` with the `SQL_ATTR_ASYNC_ENABLE` attribute to enable asynchronous calls once it has been established that there is support for asynchronous calls.
3. A call to a function that supports asynchronous execution and polling of the asynchronous function. When the application calls a function that can be run asynchronously, one of two things can happen:
 - If the function will not benefit from being run asynchronously, DB2 CLI can decide to run it synchronously and return the normal return code (other than `SQL_STILL_EXECUTING`). In this case the application runs as it would if the asynchronous mode had not been enabled.
 - DB2 CLI will perform some minimal processing (such as checking the arguments for errors), then pass the statement on to the server. Once this quick processing is complete a return code of `SQL_STILL_EXECUTING` is returned to the application.

Functions that can be called during asynchronous execution:

Once a function has been called asynchronously, only the original function, `SQLAllocHandle()`, `SQLCancel()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` can be called on the statement or the connection associated with *StatementHandle*, until the original function returns a code other than `SQL_STILL_EXECUTING`. Any other function called on *StatementHandle* or the connection associated with *StatementHandle* returns `SQL_ERROR` with an `SQLSTATE` of `HY010` (Function sequence error.).

Diagnostic information while a function is running asynchronously:

SQLGetDiagField() returns the following values when it is called on a statement handle that has an asynchronous function executing:

- The values of SQL_DIAG_CURSOR_ROW_COUNT, SQL_DIAG_DYNAMIC_FUNCTION, SQL_DIAG_DYNAMIC_FUNCTION_CODE, and SQL_DIAG_ROW_COUNT header fields are undefined.
- SQL_DIAG_NUMBER header field returns 0.
- SQL_DIAG_RETURN_CODE header field returns SQL_STILL_EXECUTING.
- All record fields return SQL_NO_DATA.

SQLGetDiagRec() always returns SQL_NO_DATA when it is called on a statement handle that has an asynchronous function executing.

Cancelling the asynchronous function call:

The application can issue a request to cancel any function that is running asynchronously by calling SQLCancel(). A function that has already finished executing cannot be cancelled.

The return code from the SQLCancel() call indicates whether the cancel request was received, not whether the execution of the asynchronous function was stopped.

The only way to tell if the function was canceled is to call it again, using the original arguments.

- If the cancel was successful, the function will return SQL_ERROR and an SQLSTATE of HY008 (Operation was cancelled.).
- If the cancel was not successful, the function will return a value other than SQL_ERROR with an SQLSTATE of HY008. For example, the function might return SQL_STILL_EXECUTING.

Related tasks:

- “Executing functions asynchronously in CLI applications” on page 129

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Header and record fields for the DiagIdentifier argument (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLCancel function (CLI) - Cancel statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetInfo function (CLI) - Get general information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnection function (CLI) - Set connection handle” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*

- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Executing functions asynchronously in CLI applications

Executing functions asynchronously in CLI applications is part of the larger task of programming with CLI. The task of enabling asynchronous functions and working with those functions involves ensuring that asynchronous execution is supported, initializing the application for asynchronous execution, and working with the functions to take advantage of asynchronous execution.

Prerequisites:

Before you begin setting up your CLI application for asynchronous execution, you must allocate an environment handle and a connection handle. This is part of the task of initializing your CLI application.

Restrictions:

An application can have at most 1 active function running in asynchronous mode on any one connection. If asynchronous mode is enabled at the connection level, all statements already allocated, as well as future statement handles allocated on the connection will be enabled for asynchronous execution.

Procedure:

1. Call `SQLGetInfo()` with *InfoType* `SQL_ASYNC_MODE` to ensure that functions can be called asynchronously. For example:

```
/* See what type of Asynchronous support is available. */
rc = SQLGetInfo( hdbc, /* Connection handle */
                SQL_ASYNC_MODE, /* Query the support available */
                &ubuffer, /* Store the result in this variable */
                4,
                &outlen);
```

The call to the `SQLGetInfo()` function will return one of the following values:

- `SQL_AM_STATEMENT`: asynchronous execution can be turned on or off at a statement level.
 - `SQL_AM_CONNECTION`: asynchronous execution can be turned on or off at a connection level.
 - `SQL_AM_NONE`: asynchronous execution is not supported. Your application cannot be set up for asynchronous execution. This will be returned for one of two reasons:
 - The datasource itself does not support asynchronous execution.
 - The DB2 CLI/ODBC configuration keyword `ASYNCENABLE` has been specifically set to disable asynchronous execution.
2. Set the `SQL_ATTR_ASYNC_ENABLE` attribute using `SQLSetStmtAttr()` or `SQLSetConnectAttr()` to enable your application for asynchronous execution if the return value from `SQLGetInfo()` is either `SQL_AM_STATEMENT` or `SQL_AM_CONNECTION`.
 - If the return value is `SQL_AM_STATEMENT`, set `SQL_ATTR_ASYNC_ENABLE` to `SQL_ASYNC_ENABLE_ON` using `SQLSetStmtAttr()`. For example:

```

/* Set statement level asynchronous execution on */
rc = SQLSetStmtAttr( hstmt, /* Statement handle */
                    SQL_ATTR_ASYNC_ENABLE,
                    (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
                    0);

```

- If the return value is `SQL_AM_CONNECTION`, set the `SQL_ATTR_ASYNC_ENABLE` to `SQL_ASYNC_ENABLE_ON` using `SQLSetConnectAttr()`. For example:

```

/* Set connection level asynchronous execution on */
rc = SQLSetConnectAttr( hstmt, /* Connection handle */
                       SQL_ATTR_ASYNC_ENABLE,
                       (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
                       0);

```

3. Call a function that supports asynchronous execution and poll the asynchronous function. Refer to the `SQL_ATTR_ASYNC_ENABLE` connection or statement attribute for a list of functions that can be executed asynchronously.

The application determines whether the function has completed by calling it repeatedly with the same arguments it used to call the function the first time. A return code of `SQL_STILL_EXECUTING` indicates it is not yet finished, any other value indicates it has completed. The value other than `SQL_STILL_EXECUTING` is the same return code it would have returned if it had executed synchronously.

The following example demonstrates a common while loop that takes both possible outcomes into account:

```

while ( (rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS) ) == SQL_STILL_EXECUTING)
{
    /* Other processing can be performed here, between each call to
    * see if SQLExecDirect() has finished running asynchronously.
    * This section will never run if CLI runs the function
    * synchronously.
    */
}
/* The application continues at this point when SQLExecDirect() */
/* has finished running. */

```

Related concepts:

- “Asynchronous execution of CLI functions” on page 127

Related tasks:

- “Initializing CLI applications” on page 18

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetInfo function (CLI) - Get general information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

- “AsyncEnable CLI/ODBC configuration keyword” on page 289

Chapter 12. Multithreaded CLI applications

Multithreaded CLI applications	133	Mixed multithreaded CLI applications	136
Application model for multithreaded CLI applications	134		

Multithreaded CLI applications

DB2 CLI supports concurrent execution of threads on the following platforms:

- AIX®
- HP-UX
- Linux
- Solaris
- Windows

On any other platform that supports threads, DB2 CLI is guaranteed to be thread safe by serializing all threaded access to the database. In other words, applications or stored procedures that use DB2 CLI can be invoked multiple times and at the same time.

Note: If you are writing applications that use DB2 CLI calls and either embedded SQL or DB2 API calls, see the documentation for multithreaded mixed applications.

Concurrent execution means that two threads can run independently of each other (on a multi-processor machine they may run simultaneously). For example, an application could implement a database-to-database copy in the following way:

- One thread connects to database A and uses `SQLExecute()` and `SQLFetch()` calls to read data from one connection into a shared application buffer.
- The other thread connects to database B and concurrently reads from the shared buffer and inserts the data into database B.

In contrast, if DB2 CLI serializes all function calls, only one thread may be executing a DB2 CLI function at a time. All other threads would have to wait until the current thread is done before it would get a chance to execute.

When to use multiple threads:

The most common reason to create another thread in a DB2 CLI application is so a thread other than the one executing can be used to call `SQLCancel()` (to cancel a long running query for example).

Most GUI-based applications use threads in order to ensure that user interaction can be handled on a higher priority thread than other application tasks. The application can simply delegate one thread to run all DB2 CLI functions (with the exception of `SQLCancel()`). In this case there are no thread-related application design issues since only one thread will be accessing the data buffers that are used to interact with DB2 CLI.

Applications that use multiple connections, and are executing statements that may take some time to execute, should consider executing DB2 CLI functions on

multiple threads to improve throughput. Such an application should follow standard practices for writing any multi-threaded application, most notably, those concerned with sharing data buffers.

Programming tips:

Any resource allocated by DB2 CLI is guaranteed to be thread-safe. This is accomplished by using either a shared global or connection specific semaphore. At any one time, only one thread can be executing a DB2 CLI function that accepts an environment handle as input. All other functions that accept a connection handle (or a statement or descriptor allocated on that connection handle) will be serialized on the connection handle.

This means that once a thread starts executing a function with a connection handle, or child of a connection handle, any other thread will block and wait for the executing thread to return. The one exception to this is `SQLCancel()`, which must be able to cancel a statement currently executing on another thread. For this reason, the most natural design is to map one thread per connection, plus one thread to handle `SQLCancel()` requests. Each thread can then execute independently of the others.

If an object is shared across threads, application timing issues may arise. For example, if a thread is using a handle in one thread, and another thread frees that handle between function calls, the next attempt to use that handle would result in a return code of `SQL_INVALID_HANDLE`.

Notes:

1. Thread safety for handles only applies for DB2 CLI applications. ODBC applications may trap since the handle in this case is a pointer and the pointer may no longer be valid if another thread has freed it. For this reason, it is best when writing an ODBC application to follow the application model for multithreaded CLI applications.
2. There may be platform or compiler specific link options required for multi-threaded applications. Refer to your compiler documentation for further details.

Related concepts:

- “Application model for multithreaded CLI applications” on page 134
- “Mixed multithreaded CLI applications” on page 136

Related reference:

- “CLI function return codes” on page 50
- “SQLCancel function (CLI) - Cancel statement” in *Call Level Interface Guide and Reference, Volume 2*

Application model for multithreaded CLI applications

The following model of a typical multithreaded CLI application is intended as an example:

- Designate a master thread which allocates:
 - m “child” threads
 - n connection handles
- Each task that requires a connection is executed by one of the child threads, and is given one of the n connections by the master thread.

- Each connection is marked as in use by the master thread until the child thread returns it to the master thread.
- Any `SQLCancel()` request is handled by the master thread.

This model allows the master thread to have more threads than connections if the threads are also used to perform non-SQL related tasks, or more connections than threads if the application wants to maintain a pool of active connections to various databases, but limit the number of active tasks.

Note: A multithreaded DB2 CLI stored procedure can only connect to the database where the stored procedure is currently executing.

Most importantly, this ensures that two threads are not trying to use the same connection handle at any one time. Although DB2 CLI controls access to its resources, the application resources such as bound columns and parameter buffers are not controlled by DB2 CLI, and the application must guarantee that a pointer to a buffer is not being used by two threads at any one time. Any deferred arguments must remain valid until the column or parameter has been unbound.

If it is necessary for two threads to share a data buffer, the application must implement some form of synchronization mechanism. For example, in the database-to-database copy scenario where one thread connects to database A and reads data from one connection into a shared application buffer while the other thread connects to database B and concurrently reads from the shared buffer and inserts data into database B, the use of the shared buffer must be synchronized by the application.

Application deadlocks:

The application must be aware of the possibility of creating deadlock situations with shared resources in the database and the application.

DB2 can detect deadlocks at the server and rollback one or more transactions to resolve them. An application may still deadlock if:

- two threads are connected to the same database, and
- one thread is holding an application resource 'A' and is waiting for a database resource 'B', and
- the other thread has a lock on the database resource 'B' while waiting for the application resource 'A'.

In this case the DB2 server is only going to see a lock, not a deadlock, and unless the database `LockTimeout` configuration keyword is set, the application will wait forever.

The model suggested above avoids this problem by not sharing application resources between threads once a thread starts executing on a connection.

Related concepts:

- “Mixed multithreaded CLI applications” on page 136
- “Multithreaded CLI applications” on page 133

Related reference:

- “locktimeout - Lock timeout configuration parameter” in *Performance Guide*
- “SQLCancel function (CLI) - Cancel statement” in *Call Level Interface Guide and Reference, Volume 2*

Mixed multithreaded CLI applications

It is possible for a multi-threaded application to mix CLI calls with DB2 API calls and embedded SQL. The type of the call executed earliest in the application determines the best way to organize the application:

DB2 CLI Calls first

The DB2 CLI driver automatically calls the DB2 context APIs to allocate and manage contexts for the application. This means that any application that calls `SQLAllocEnv()` before calling any other DB2 API or embedded SQL will be initialized with the context type set to `SQL_CTX_MULTI_MANUAL`.

In this case the application should allow DB2 CLI to allocate and manage all contexts. Use DB2 CLI to allocate all connection handles and to perform all connections. Call the `SQLSetConnect()` function in each thread prior to calling any embedded SQL. DB2 APIs can be called after any DB2 CLI function has been called in the same thread.

DB2 API or embedded SQL calls first

The DB2 CLI driver does not automatically call the DB2 context APIs if the application calls any DB2 API or embedded SQL functions before a CLI function.

This means that any thread that calls a DB2 API or embedded SQL function must be attached to a context, otherwise the call will fail with an SQLCODE of `SQL1445N`. This can be done by calling the DB2 API `sqlAttachToCtx()` which will explicitly attach the thread to a context, or by calling any DB2 CLI function (`SQLSetConnection()` for example). In this case, the application must explicitly manage all contexts.

Use the context APIs to allocate and attach to contexts prior to calling DB2 CLI functions (`SQLAllocEnv()` will use the existing context as the default context). Use the `SQL_ATTR_CONN_CONTEXT` connection attribute to explicitly set the context that each DB2 CLI connection should use.

Note: It is recommended that you do not use the default application stack size, but instead increase the stack size to at least 256 000. DB2 requires a minimum application stack size of 256 000 when calling a DB2 function. You must ensure therefore, that you allocate a total stack size that is large enough for both your application and the minimum requirements for a DB2 function call.

Related concepts:

- “Application model for multithreaded CLI applications” on page 134
- “Multithreaded CLI applications” on page 133
- “DB2 Call Level Interface (CLI) versus embedded SQL” on page 4

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLAllocEnv function (CLI) - Allocate environment handle” in *Call Level Interface Guide and Reference, Volume 2*
- “sqlAttachToCtx API - Attach to context” in *Administrative API Reference*

Chapter 13. Multisite updates (two phase commit)

Multisite updates (two phase commit) in CLI applications	137	Process-based XA-compliant Transaction Program Monitor (XA TP) programming considerations for CLI applications	141
DB2 as transaction manager in CLI applications	138		

Multisite updates (two phase commit) in CLI applications

A typical transaction scenario portrays an application which interacts with only one database server in a transaction. Even though concurrent connections allow for concurrent transactions, the different transactions are not coordinated.

With multisite updates, the two phase commit (2PC) protocol, and coordinated distributed transactions, an application is able to update data in multiple remote database servers with guaranteed integrity.

Note: Multisite update is also known as Distributed Unit of Work (DUOW).

A typical banking transaction is a good example of a multisite update. Consider the transfer of money from one account to another in a different database server. In such a transaction it is critical that the updates that implement the debit operation on one account do not get committed unless the updates required to process the credit to the other account are committed as well. Multisite update considerations apply when data representing these accounts is managed by two different database servers

Some multisite updates involve the use of a transaction manager (TM) to coordinate two-phase commit among multiple databases. DB2 CLI applications can be written to use various transaction managers:

- DB2 as transaction manager
- Process-based XA-compliant transaction program monitor
- Host and AS/400® database servers

Note: There is no specific DB2 CLI/ODBC client configuration required when connecting to a host or iSeries™ database server, although the machine running DB2 Connect™ might require certain configuration settings to enable running multisite update mode against the host.

Related concepts:

- “Configuration parameter settings for multisite update applications” in *SQL Guide*
- “DB2 as transaction manager in CLI applications” on page 138
- “Multisite Updates” in *DB2 Connect User’s Guide*
- “Process-based XA-compliant Transaction Program Monitor (XA TP) programming considerations for CLI applications” on page 141

Related tasks:

- “Enabling Multisite Updates using the Control Center” in *DB2 Connect User’s Guide*

DB2 as transaction manager in CLI applications

Configuration of DB2 as transaction manager:

DB2 CLI/ODBC applications can use DB2 itself as the Transaction Manager (DB2 TM) to coordinate distributed transactions against all IBM database servers.

The DB2 Transaction Manager must be set up according to the information in the DB2 transaction manager configuration documentation.

To use DB2 as the transaction manager in CLI/ODBC applications, the following configurations must be applied:

- The `SQL_ATTR_CONNECTTYPE` environment attribute must be set. This attribute controls whether the application is to operate in a coordinated or uncoordinated distributed environment. Commits or rollbacks among multiple database connections are coordinated in a coordinated distributed environment. The two possible values for this attribute are:
 - `SQL_CONCURRENT_TRANS` - supports single database per transaction semantics. Multiple concurrent connections to the same database and to different databases are permitted. Each connection has its own commit scope. No effort is made to enforce coordination of transactions. This is the default and corresponds to a Type 1 `CONNECT` in embedded SQL.
 - `SQL_COORDINATED_TRANS` - supports multiple databases per transaction semantics. A coordinated transaction is one in which commits or rollbacks among multiple database connections are coordinated. Setting `SQL_ATTR_CONNECTTYPE` to this value corresponds to Type 2 `CONNECT` in embedded SQL.

It is recommended that the application set this environment attribute with a call to `SQLSetEnvAttr()`, if necessary, as soon as the environment handle has been allocated. However, since ODBC applications cannot access `SQLSetEnvAttr()`, they must set this using `SQLSetConnectAttr()` after each connection handle is allocated, but before any connections have been established.

All connections on an environment handle must have the same `SQL_ATTR_CONNECTTYPE` setting. An environment cannot have a mixture of concurrent and coordinated connections. The type of the first connection will determine the type of all subsequent connections. `SQLSetEnvAttr()` will return an error if an application attempts to change the connect type while there is an active connection.

- If `SQL_ATTR_CONNECTTYPE` is set to `SQL_COORDINATED_TRANS` as described above, two-phase commit is used to commit the work done by each database in a multiple database transaction. This requires the use of a Transaction Manager to coordinate two-phase commits amongst the databases that support this protocol. Multiple readers and multiple updaters are allowed within a transaction.
- The function `SQLEndTran()` must be used in a multisite update environment when DB2 is acting as the transaction manager.

Application flows in concurrent and coordinated transactions:

Figure 9 on page 139 shows the logical flow of an application executing statements on two `SQL_CONCURRENT_TRANS` connections ('A' and 'B'), and indicates the scope of the transactions.

Figure 10 on page 140 shows the same statements being executed on two SQL_COORDINATED_TRANS connections ('A' and 'B'), and the scope of a coordinated distributed transaction.

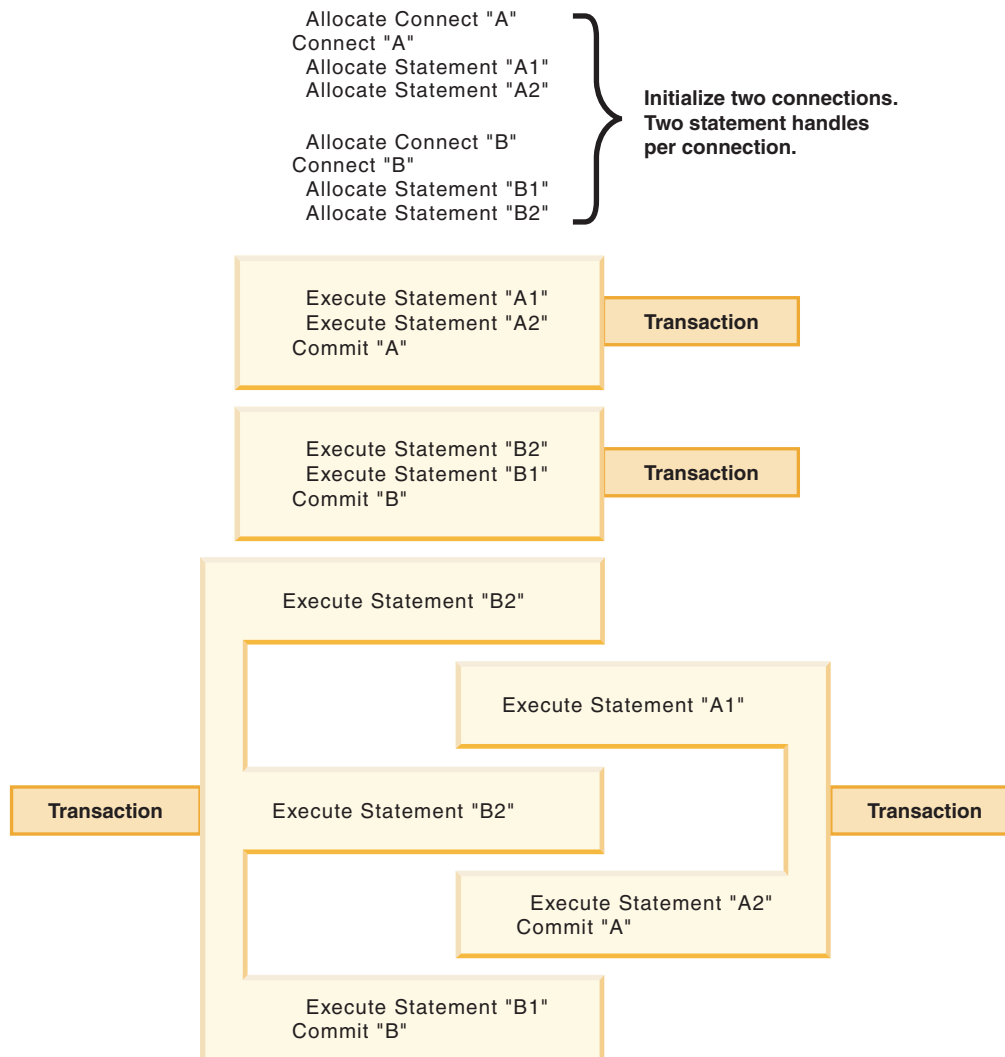


Figure 9. Multiple connections with concurrent transactions

Allocate Environment
 Set Environment Attribute
 (SQL_ATTR_CONNECTTYPE)

Allocate Connect "A"
 Connect "A"
 (SQL_COORDINATED_TRANS)

Allocate Statement "A1"
 Allocate Statement "A2"

Allocate Connect "B"
 Connect "B"
 (SQL_COORDINATED_TRANS)

Allocate Statement "B1"
 Allocate Statement "B2"

**Initialize two connections.
 Two statement handles
 per connection.**

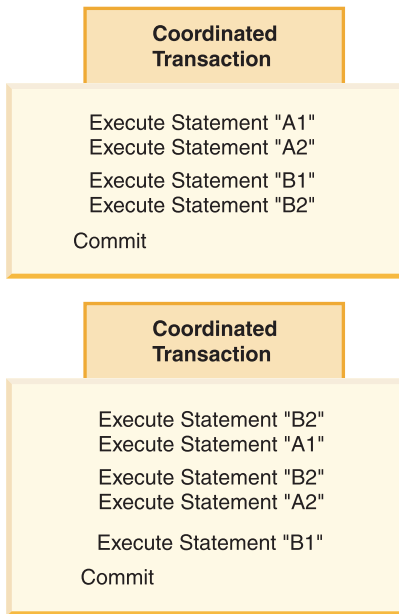


Figure 10. Multiple connections with coordinated transactions

Restrictions:

Mixing embedded SQL and CLI/ODBC calls in a multisite update environment is supported, but all the same restrictions of writing mixed applications are imposed.

Related concepts:

- "DB2 Database transaction manager configuration" in *Administration Guide: Planning*
- "Considerations for mixing embedded SQL and DB2 CLI" on page 183
- "db2cli.ini initialization file" on page 277
- "Multisite updates (two phase commit) in CLI applications" on page 137

Related reference:

- "CONNECT (Type 1) statement" in *SQL Reference, Volume 2*
- "CONNECT (Type 2) statement" in *SQL Reference, Volume 2*
- "ConnectType CLI/ODBC configuration keyword" on page 301

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLEndTran function (CLI) - End transactions of a connection or an Environment” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetEnvAttr function (CLI) - Set environment attribute” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbmcon.c -- How to use multiple databases”
- “dbmconx.c -- How to use multiple databases with embedded SQL.”

Process-based XA-compliant Transaction Program Monitor (XA TP) programming considerations for CLI applications

Process-based XA TPs, such as CICS[®] and Encina[®], start up one application server per process. In each application-server process, the connections are already established using the XA API (`xa_open`). This section describes the environment configurations and considerations for running DB2 CLI/ODBC applications under this environment.

Configuration:

The XA Transaction Manager must be set up according to the configuration considerations for XA transaction managers.

Note: Setting the CLI/ODBC configuration keywords for connections is no longer required when in an XA Transactional processing environment.

Programming considerations:

DB2 CLI/ODBC applications written for this environment must complete the following steps:

- The application must first call `SQLConnect()` or `SQLDriverConnect()` to associate the TM-opened connections with the CLI/ODBC connection handle. The data source name must be specified. User ID and Password are optional.
- The application must call the XA TM to do a commit or rollback. As a result, since the CLI/ODBC driver does not know that the transaction has ended, the application should do the following before exiting:
 - Drop all CLI/ODBC statement handles.
 - Free up the connection handle by calling `SQLDisconnect()` and `SQLFreeHandle()`. The actual database connection will not be disconnected until the XA TM performs an `xa_close`.

Restrictions:

Mixing embedded SQL and CLI/ODBC calls in a multisite update environment is supported, but all the same restrictions of writing mixed applications are imposed.

Related concepts:

- “Configuration considerations for XA transaction managers” in *Administration Guide: Planning*
- “Considerations for mixing embedded SQL and DB2 CLI” on page 183
- “DB2 as transaction manager in CLI applications” on page 138
- “Multisite updates (two phase commit) in CLI applications” on page 137

Related reference:

- “SQLConnect function (CLI) - Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDisconnect function (CLI) - Disconnect from a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*

Chapter 14. Unicode

Unicode CLI applications	143	Unicode function calls to ODBC driver managers	145
Unicode functions (CLI)	144		

Unicode CLI applications

There are two main areas of support for DB2 CLI Unicode applications:

- The addition of a set of functions that accept Unicode string arguments in place of ANSI string arguments.
- The addition of new C and SQL data types to describe Unicode data.

To be considered a Unicode application, the application must connect to the database using either `SQLConnectW()` or `SQLDriverConnectW()`. This will ensure that CLI will consider Unicode the preferred method of communication between itself and the database.

ODBC adds types to the set of C and SQL types that already exist to accommodate Unicode, and CLI uses these additional types accordingly. The new C type, `SQL_C_WCHAR`, indicates that the C buffer contains Unicode data. The DB2 CLI/ODBC driver considers all Unicode data exchanged with the application to be UCS-2 in native-endian format. The new SQL types, `SQL_WCHAR`, `SQL_WVARCHAR`, and `SQL_WLONGVARCHAR`, indicate that a particular column or parameter marker contains Unicode data. For DB2 Unicode databases, graphic columns are described using the new types. Conversion is allowed between `SQL_C_WCHAR` and `SQL_CHAR`, `SQL_VARCHAR`, `SQL_LONGVARCHAR` and `SQL_CLOB`, as well as with the graphic data types.

Note: UCS-2 is a fixed-length character encoding scheme that uses 2 bytes to represent each character. When referring to the number of characters in a UCS-2 encoded string, the count is simply the number of `SQLWCHAR` elements needed to store the string.

Obsolete CLI/ODBC keyword values:

Before Unicode applications were supported, applications that were written to work with single-byte character data could be made to work with double-byte graphic data by a series of DB2 CLI configuration keywords, such as `Graphic=1,2` or `3`, `Patch2=7`. These workarounds presented graphic data as character data, and also affected the reported length of the data. These keywords are no longer required for Unicode applications, and should not be used due to the risk of potential side effects. If it is not known if a particular application is a Unicode application, try without any of the keywords that affect the handling of graphic data.

Literals in unicode databases:

In non-Unicode databases, data in `LONG VARGRAPHIC` and `LONG VARCHAR` columns cannot be compared. Data in `GRAPHIC/VARGRAPHIC` and `CHAR/VARCHAR` columns can only be compared, or assigned to each other, using explicit cast functions since no implicit code page conversion is supported. This includes `GRAPHIC/VARGRAPHIC` and `CHAR/VARCHAR` literals where a

GRAPHIC/VARGRAPHIC literal is differentiated from a CHAR/VARCHAR literal by a G prefix. For Unicode databases, casting between GRAPHIC/VARGRAPHIC and CHAR/VARCHAR literals is not required. Also, a G prefix is not required in front of a GRAPHIC/VARGRAPHIC literal. Provided at least one of the arguments is a literal, implicit conversions occur. This allows literals with or without the G prefix to be used within statements that use either `SQLPrepareW()` or `SQLExecDirect()`. Literals for LONG VARGRAPHICs still must have a G prefix.

Related concepts:

- “Unicode function calls to ODBC driver managers” on page 145
- “Unicode functions (CLI)” on page 144
- “Applications connected to Unicode databases” in *SQL Guide*

Related reference:

- “C data types for CLI applications” on page 43
- “Patch2 CLI/ODBC configuration keyword” on page 352
- “SQL symbolic and default data types for CLI applications” on page 41

Unicode functions (CLI)

DB2 CLI Unicode functions accept Unicode string arguments in place of ANSI string arguments. The Unicode string arguments must be in UCS-2 encoding (native-endian format). ODBC API functions have suffixes to indicate the format of their string arguments: those that accept Unicode end in *W*, and those that accept ANSI have no suffix (ODBC adds equivalent functions with names that end in *A*, but these are not offered by DB2 CLI). The following is a list of functions available in DB2 CLI which have both ANSI and Unicode versions:

SQLBrowseConnect	SQLForeignKeys	SQLPrimaryKeys
SQLColAttribute	SQLGetConnectAttr	SQLProcedureColumns
SQLColAttributes	SQLGetConnectOption	SQLProcedures
SQLColumnPrivileges	SQLGetCursorName	SQLSetConnectAttr
SQLColumns	SQLGetDescField	SQLSetConnectOption
SQLConnect	SQLGetDescRec	SQLSetCursorName
SQLDataSources	SQLGetDiagField	SQLSetDescField
SQLDescribeCol	SQLGetDiagRec	SQLSetStmtAttr
SQLDriverConnect	SQLGetInfo	SQLSpecialColumns
SQLError	SQLGetStmtAttr	SQLStatistics
SQLExecDirect	SQLNativeSQL	SQLTablePrivileges
SQLExtendedPrepare	SQLPrepare	SQLTables

Unicode functions that have arguments which are always the length of strings interpret these arguments as the number of `SQLWCHAR` elements needed to store the string. For functions that return length information for server data, the display size and precision are again described in terms of the number of `SQLWCHAR` elements used to store them. When the length (transfer size of the data) could refer to string or non-string data, it is interpreted as the number of bytes needed to store the data.

For example, `SQLGetInfoW()` will still take the length as the number of bytes, but `SQLExecDirectW()` will use the number of `SQLWCHAR` elements. Consider a single character from the UTF-16 extended character set (UTF-16 is an extended character set of UCS-2; Microsoft Windows 2000 and Microsoft Windows XP use UTF-16). Microsoft Windows 2000 will use two `SQL_C_WCHAR` elements, which is equivalent to 4 bytes, to store this single character. The character therefore has a display size of 1, a string length of 2 (when using `SQL_C_WCHAR`), and a byte

count of 4. CLI will return data from result sets in either Unicode or ANSI, depending on the application's binding. If an application binds to SQL_C_CHAR, the driver will convert SQL_WCHAR data to SQL_CHAR. An ODBC driver manager, if used, maps SQL_C_WCHAR to SQL_C_CHAR for ANSI drivers but does no mapping for Unicode drivers.

ANSI to Unicode function mappings:

The syntax for a DB2 CLI Unicode function is the same as the syntax for its corresponding ANSI function, except that SQLCHAR parameters are defined as SQLWCHAR. Character buffers defined as SQLPOINTER in the ANSI syntax can be defined as either SQLCHAR or SQLWCHAR in the Unicode function. Refer to the ANSI version of the CLI Unicode functions for ANSI syntax details.

Related concepts:

- “Unicode function calls to ODBC driver managers” on page 145
- “Unicode CLI applications” on page 143

Related reference:

- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*
- “C data types for CLI applications” on page 43
- “SQL symbolic and default data types for CLI applications” on page 41

Unicode function calls to ODBC driver managers

ODBC-compliant applications can access a DB2 database through the DB2 CLI/ODBC driver in one of two ways: linking to the DB2 CLI/ODBC driver library or linking to the ODBC driver manager library. This topic discusses CLI applications that link to the ODBC driver manager library.

- Direct access - An application links to the DB2 CLI/ODBC driver library and makes calls to exported CLI/ODBC functions. Unicode applications accessing the DB2 CLI/ODBC driver directly should access and perform transactions against the database using the CLI Unicode functions, and use SQLWCHAR buffers with the understanding that all Unicode data is UCS-2. To identify itself as a Unicode application, the application must connect to the database using either `SQLConnectW()` or `SQLDriverConnectW()`.
- Indirect access - An application links to an ODBC driver manager library and makes calls to standard ODBC functions. The ODBC driver manager then loads the DB2 CLI/ODBC driver and calls exported ODBC functions on behalf of the application. The data passed to the DB2 CLI/ODBC driver from the application might be converted by the ODBC driver manager. An application identifies itself to an ODBC driver manager as a Unicode application by calling `SQLConnectW()` or `SQLDriverConnectW()`.

When connecting to a data source, the ODBC driver manager checks to see if the requested driver exports the `SQLConnectW()` function. If the function is supported, the ODBC driver is considered a Unicode driver, and all subsequent calls in the application to ODBC functions are routed to the functions' Unicode equivalents (identified by the 'W' suffix; for example, `SQLConnectW()`) by the ODBC driver manager. If the application calls Unicode functions, no string conversion is necessary, and the ODBC driver manager calls the Unicode functions directly. If the application calls ANSI functions, the ODBC driver manager converts all ANSI strings to Unicode strings prior to calling the equivalent Unicode function.

If an application calls Unicode functions, but the driver does not export `SQLConnectW()`, then the ODBC driver manager routes any Unicode function calls to their ANSI equivalents. All Unicode strings are converted by the ODBC driver manager to ANSI strings in the application's code page before calling the equivalent ANSI function. This might result in data loss if the application uses Unicode characters which cannot be converted to the application's code page.

Various ODBC driver managers use different encoding schemes for Unicode strings, depending on the operating system:

Table 11. Unicode string encoding schemes by operating system

Driver manager	Operating system	
	Microsoft Windows	Linux and UNIX
Microsoft ODBC Driver Manager	UTF-16*	not applicable
unixODBC Driver Manager	UCS-2	UCS-2
DataDirect Connect for ODBC Driver Manager	UTF-16*	UTF-8

* UTF-16 is a superset of UCS-2 and therefore is compatible

DataDirect Connect for ODBC Driver Manager UNIX restrictions:

Complications arise when using the DB2 CLI/ODBC driver with the DataDirect Connect for ODBC Driver Manager in the UNIX environment because of the use of UTF-8 character encoding by the driver manager. UTF-8 is a variable length character encoding scheme using anywhere from 1 to 6 bytes to store characters. UTF-8 and UCS-2 are not inherently compatible, and passing UTF-8 data to the DB2 CLI/ODBC driver (which expects UCS-2) might result in application errors, data corruption, or application exceptions.

To avoid this problem, the DataDirect Connect for ODBC Driver Manager 4.2 Service Pack 2 recognizes a DB2 CLI/ODBC driver and not use the Unicode functions, effectively treating the DB2 CLI/ODBC driver as an ANSI-only driver. Before release 4.2 Service Pack 2, the DataDirect Connect for ODBC Driver Manager had to be linked with the "_36" version of the DB2 CLI/ODBC driver which does not export the `SQLConnectW()` function.

Related concepts:

- "Comparison of DB2 CLI and Microsoft ODBC" on page 9
- "Unicode CLI applications" on page 143
- "Unicode functions (CLI)" on page 144

Related tasks:

- "Setting up the unixODBC Driver Manager" on page 212
- "Setting up the UNIX ODBC environment" on page 210

Chapter 15. User-defined types (UDT)

Distinct type usage in CLI applications

In addition to SQL data types (referred to as *base* SQL data types), new distinct types can be defined by the user. This variety of user defined types (UDTs) shares its internal representation with an existing type, but is considered to be a separate and incompatible type for most operations. Distinct types are created using the CREATE DISTINCT TYPE SQL statement.

Distinct types help provide the strong typing control needed in object oriented programming by ensuring that only those functions and operators explicitly defined on a distinct type can be applied to its instances. Applications continue to work with C data types for application variables, and only need to consider the distinct types when constructing SQL statements.

This means:

- All SQL to C data type conversion rules that apply to the built-in type apply to distinct types.
- Distinct types will have the same default C Type as the built-in type.
- `SQLDescribeCol()` will return the built-in type information. The user defined type name can be obtained by calling `SQLColAttribute()` with the input descriptor type set to `SQL_DESC_DISTINCT_TYPE`.
- SQL predicates that involve parameter markers must be explicitly cast to the distinct type. This is required since the application can only deal with the built-in types, so before any operation can be performed using the parameter, it must be cast from the C built-in type to the distinct type; otherwise an error will occur when the statement is prepared.

Related concepts:

- “User-defined type (UDT) usage in CLI applications” on page 148

Related reference:

- “C data types for CLI applications” on page 43
- “CREATE DISTINCT TYPE statement” in *SQL Reference, Volume 2*
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQL to C data conversion in CLI” on page 399
- “SQLColAttribute function (CLI) - Return a column attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtudt.c -- How to create, use, and drop user-defined distinct types.”

User-defined type (UDT) usage in CLI applications

User-defined types (UDTs) are database types defined by the user to provide structure or strong typing not available with conventional SQL types. There are three varieties of UDT: distinct types, structured types, and reference types.

A CLI application may want to determine whether a given database column is a UDT, and if so, the variety of UDT. The descriptor field `SQL_DESC_USER_DEFINED_TYPE_CODE` may be used to obtain this information. When `SQL_DESC_USER_DEFINED_TYPE_CODE` is retrieved using `SQLColAttribute()` or directly from the IPD using `SQLGetDescField()`, it will have one of the following numeric values:

```
SQL_TYPE_BASE (this is a regular SQL type, not a UDT)
SQL_TYPE_DISTINCT (this value indicates that the column
                  is a distinct type)
SQL_TYPE_STRUCTURED (this value indicates that the column
                   is a structured type)
SQL_TYPE_REFERENCE (this value indicates that the column
                   is a reference type)
```

Additionally, the following descriptor fields may be used to obtain the type names:

- `SQL_DESC_REFERENCE_TYPE` contains the name of the reference type, or an empty string if the column is not a reference type.
- `SQL_DESC_STRUCTURED_TYPE` contains the name of the structured type, or an empty string if the column is not a structured type.
- `SQL_DESC_USER_TYPE` or `SQL_DESC_DISTINCT_TYPE` contains the name of the distinct type, or an empty string if the column is not a distinct type.

The descriptor fields listed above return the schema as part of the name. If the schema is less than 8 letters, it is padded with blanks.

The connection attribute `SQL_ATTR_TRANSFORM_GROUP` allows an application to set the transform group, and is an alternative to the SQL statement `SET CURRENT DEFAULT TRANSFORM GROUP`.

A CLI application may not wish to repeatedly obtain the value of the `SQL_DESC_USER_DEFINED_TYPE_CODE` descriptor field to determine if columns contain UDTs. For this reason, there is an attribute called `SQL_ATTR_RETURN_USER_DEFINED_TYPES` at both the connection and the statement handle level. When set to `SQL_TRUE` using `SQLSetConnectAttr()`, CLI returns `SQL_DESC_USER_DEFINED_TYPE` where you would normally find SQL types in results from calls to `SQLColAttribute()`, `SQLDescribeCol()` and `SQLGetDescField()`. This allows the application to check for this special type, and then do special processing for UDTs. The default value for this attribute is `SQL_FALSE`.

When the `SQL_ATTR_RETURN_USER_DEFINED_TYPES` attribute is set to `SQL_TRUE`, the descriptor field `SQL_DESC_TYPE` will no longer return the "base" SQL type of the UDT, that is, the SQL type that the UDT is based on or transforms to. For this reason, the descriptor field `SQL_DESC_BASE_TYPE` will always return the base type of UDTs, and the SQL type of normal columns. This field simplifies modules of a program that do not deal specifically with UDTs that would otherwise have to change the connection attribute.

Note that `SQLBindParameter()` will not allow you to bind a parameter of the type `SQL_USER_DEFINED_TYPE`. You must still bind parameters using the base SQL

type, which you can obtain using the descriptor field `SQL_DESC_BASE_TYPE`. For example, here is the `SQLBindParameter()` call used when binding to a column with a distinct type based on `SQL_VARCHAR`:

```
sqlrc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,  
                          SQL_VARCHAR, 30, 0, &c2, 30, NULL);
```

Related concepts:

- “Distinct type usage in CLI applications” on page 147

Related reference:

- “CREATE DISTINCT TYPE statement” in *SQL Reference, Volume 2*
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLColAttribute function (CLI) - Return a column attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dtudt.c -- How to create, use, and drop user-defined distinct types.”
- “udfcli.c -- How to work with different types of user-defined functions (UDFs)”

Chapter 16. Descriptors

Descriptors in CLI applications	151	Descriptor manipulation with descriptor handles in CLI applications	158
Consistency checks for descriptors in CLI applications	154	Descriptor manipulation without using descriptor handles in CLI applications.	160
Descriptor allocation and freeing	155		

Descriptors in CLI applications

DB2 CLI stores information (data types, size, pointers, and so on) about columns in a result set, and parameters in an SQL statement. The bindings of application buffers to columns and parameters must also be stored. *Descriptors* are a logical view of this information, and provide a way for applications to query and update this information.

Many CLI functions make use of descriptors, but the application itself does not need to manipulate them directly.

For instance:

- When an application binds column data using `SQLBindCol()`, descriptor fields are set that completely describe the binding.
- A number of statement attributes correspond to the header fields of a descriptor. In this case you can achieve the same effect calling `SQLSetStmtAttr()` as calling the corresponding function `SQLSetDescField()` that sets the values in the descriptor directly.

Although no database operations require direct access to descriptors, there are situations where working directly with the descriptors will be more efficient or result in simpler code. For instance, a descriptor that describes a row fetched from a table can then be used to describe a row inserted back into the table.

There are four types of descriptors:

Application Parameter Descriptor (APD)

Describes the application buffers (pointers, data types, scale, precision, length, maximum buffer length, and so on) that are bound to parameters in an SQL statement. If the parameters are part of a CALL statement they may be input, output, or both. This information is described using the application's C data types.

Application Row Descriptor (ARD)

Describes the application buffers bound to the columns. The application may specify different data types from those in the implementation row descriptor (IRD) to achieve data conversion of column data. This descriptor reflects any data conversion that the application may specify.

Implementation Parameter Descriptor (IPD)

Describes the parameters in the SQL statement (SQL type, size, precision, and so on).

- If the parameter is used as input, this describes the SQL data that the database server will receive after DB2 CLI has performed any required conversion.

- If the parameter is used as output, this describes the SQL data before DB2 CLI performs any required conversion to the application's C data types.

Implementation Row Descriptor (IRD)

Describes the row of data from the result set before DB2 CLI performs any required data conversion to the application's C data types.

The only difference between the four types of descriptors described above is how they are used. One of the benefits of descriptors is that a single descriptor can be used to serve multiple purposes. For instance, a row descriptor in one statement can be used as a parameter descriptor in another statement.

As soon as a descriptor exists, it is either an application descriptor or an implementation descriptor. This is the case even if the descriptor has not yet been used in a database operation. If the descriptor is allocated by the application using `SQLAllocHandle()` then it is an application descriptor.

Values stored in a descriptor:

Each descriptor contains both header fields and record fields. These fields together completely describe the column or parameter.

Header fields:

Each header field occurs once in each descriptor. Changing one of these fields affects all columns or parameters.

Many of the following header fields correspond to a statement attribute. Setting the header field of the descriptor using `SQLSetDescField()` is the same as setting the corresponding statement attribute using `SQLSetStmtAttr()`. The same holds true for retrieving the information using `SQLGetDescField()` or `SQLGetStmtAttr()`. If your application does not already have a descriptor handle allocated then it is more efficient to use the statement attribute calls instead of allocating the descriptor handle, and then using the descriptor calls.

The following is a list of the header fields:

<code>SQL_DESC_ALLOC_TYPE</code>	<code>SQL_DESC_BIND_TYPE^a</code>
<code>SQL_DESC_ARRAY_SIZE^a</code>	<code>SQL_DESC_COUNT</code>
<code>SQL_DESC_ARRAY_STATUS_PTR^a</code>	<code>SQL_DESC_ROWS_PROCESSED_PTR^a</code>
<code>SQL_DESC_BIND_OFFSET_PTR^a</code>	

Note:

^a This header field corresponds to a statement attribute.

The descriptor header field `SQL_DESC_COUNT` is the one-based index of the highest-numbered descriptor record that contains information (and not a count of the number of columns or parameters). DB2 CLI automatically updates this field (and the physical size of the descriptor) as columns or parameters are bound and unbound. The initial value of `SQL_DESC_COUNT` is 0 when a descriptor is first allocated.

Descriptor records:

Zero or more descriptor records are contained in a single descriptor. As new columns or parameters are bound, new descriptor records are added to the descriptor. When a column or parameter is unbound, the descriptor record is removed.

The following is a list of the fields in a descriptor record. They describe a column or parameter, and occur once in each descriptor record.

SQL_DESC_AUTO_UNIQUE_VALUE	SQL_DESC_LOCAL_TYPE_NAME
SQL_DESC_BASE_COLUMN_NAME	SQL_DESC_NAME
SQL_DESC_BASE_TABLE_NAME	SQL_DESC_NULLABLE
SQL_DESC_CASE_SENSITIVE	SQL_DESC_OCTET_LENGTH
SQL_DESC_CATALOG_NAME	SQL_DESC_OCTET_LENGTH_PTR
SQL_DESC_CONCISE_TYPE	SQL_DESC_PARAMETER_TYPE
SQL_DESC_DATA_PTR	SQL_DESC_PRECISION
SQL_DESC_DATETIME_INTERVAL_CODE	SQL_DESC_SCALE
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQL_DESC_SCHEMA_NAME
SQL_DESC_DISPLAY_SIZE	SQL_DESC_SEARCHABLE
SQL_DESC_FIXED_PREC_SCALE	SQL_DESC_TABLE_NAME
SQL_DESC_INDICATOR_PTR	SQL_DESC_TYPE
SQL_DESC_LABEL	SQL_DESC_TYPE_NAME
SQL_DESC_LENGTH	SQL_DESC_UNNAMED
SQL_DESC_LITERAL_PREFIX	SQL_DESC_UNSIGNED
SQL_DESC_LITERAL_SUFFIX	SQL_DESC_UPDATABLE

Deferred fields:

Deferred fields are created when the descriptor header or a descriptor record is created. The addresses of the defined variables are stored but not used until a later point in the application. The application must not deallocate or discard these variables between the time it associates them with the fields and the time CLI reads or writes them.

The following table lists the deferred fields and the meaning of a null pointer where applicable:

Table 12. Deferred fields

Field	Meaning of Null value
SQL_DESC_DATA_PTR	The record is unbound.
SQL_DESC_INDICATOR_PTR	(none)
SQL_DESC_OCTET_LENGTH_PTR (ARD and APD only)	<ul style="list-style-type: none"> • ARD: The length information for that column is not returned. • APD: If the parameter is a character string, the driver assumes that string is null-terminated. For output parameters, a null value in this field prevents the driver from returning length information. (If the SQL_DESC_TYPE field does not indicate a character-string parameter, the SQL_DESC_OCTET_LENGTH_PTR field is ignored.)
SQL_DESC_ARRAY_STATUS_PTR (multirow fetch only)	A multirow fetch failed to return this component of the per-row diagnostic information.
SQL_DESC_ROWS_PROCESSED_PTR (multirow fetch only)	(none)

Bound descriptor records:

The `SQL_DESC_DATA_PTR` field in each descriptor record points to a variable that contains the parameter value (for APDs) or the column value (for ARDs). This is a deferred field that defaults to null. Once the column or parameter is bound it points to the parameter or column value. At this point the descriptor record is said to be bound.

Application Parameter Descriptors (APD)

Each bound record constitutes a bound parameter. The application must bind a parameter for each input and output parameter marker in the SQL statement before the statement is executed.

Application Row Descriptors (ARD)

Each bound record relates to a bound column.

Related concepts:

- “Consistency checks for descriptors in CLI applications” on page 154
- “Descriptor allocation and freeing” on page 155
- “Descriptor manipulation with descriptor handles in CLI applications” on page 158
- “Descriptor manipulation without using descriptor handles in CLI applications” on page 160

Related reference:

- “Descriptor FieldIdentifier argument values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “Descriptor header and record field initialization values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescField function (CLI) - Set a single field of a descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Consistency checks for descriptors in CLI applications

A consistency check is performed automatically whenever an application sets the `SQL_DESC_DATA_PTR` field of the APD or ARD. The check ensures that various fields are consistent with each other, and that appropriate data types have been specified. Calling `SQLSetDescRec()` always prompts a consistency check. If any of the fields is inconsistent with other fields, `SQLSetDescRec()` will return `SQLSTATE HY021`, “Inconsistent descriptor information.”

To force a consistency check of IPD fields, the application can set the `SQL_DESC_DATA_PTR` field of the IPD. This setting is only used to force the consistency check. The value is not stored and cannot be retrieved by a call to `SQLGetDescField()` or `SQLGetDescRec()`.

A consistency check cannot be performed on an IRD.

Application descriptors:

Whenever an application sets the `SQL_DESC_DATA_PTR` field of an APD, ARD, or IPD, DB2 CLI checks that the value of the `SQL_DESC_TYPE` field and the values applicable to that `SQL_DESC_TYPE` field are valid and consistent. This check is always performed when `SQLBindParameter()` or `SQLBindCol()` is called, or when `SQLSetDescRec()` is called for an APD, ARD, or IPD. This consistency check includes the following checks on application descriptor fields:

- The `SQL_DESC_TYPE` field must be one of the valid C or SQL types. The `SQL_DESC_CONCISE_TYPE` field must be one of the valid C or SQL types.
- If the `SQL_DESC_TYPE` field indicates a numeric type, the `SQL_DESC_PRECISION` and `SQL_DESC_SCALE` fields are verified to be valid.
- If the `SQL_DESC_CONCISE_TYPE` field is a time data type the `SQL_DESC_PRECISION` field is verified to be a valid seconds precision.

The `SQL_DESC_DATA_PTR` field of an IPD is not normally set; however, an application can do so to force a consistency check of IPD fields. A consistency check cannot be performed on an IRD. The value that the `SQL_DESC_DATA_PTR` field of the IPD is set to is not actually stored, and cannot be retrieved by a call to `SQLGetDescField()` or `SQLGetDescRec()`; the setting is made only to force the consistency check.

Related concepts:

- “Descriptors in CLI applications” on page 151

Related reference:

- “Descriptor FieldIdentifier argument values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “Descriptor header and record field initialization values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data” in *Call Level Interface Guide and Reference, Volume 2*

Descriptor allocation and freeing

Descriptors are allocated in one of two ways:

Implicitly allocated descriptors

When a statement handle is allocated, a set of four descriptors are implicitly allocated. When the statement handle is freed, all implicitly allocated descriptors on that handle are freed as well.

To obtain handles to these implicitly allocated descriptors an application can call `SQLGetStmtAttr()`, passing the statement handle and an *Attribute* value of:

- `SQL_ATTR_APP_PARAM_DESC` (APD)
- `SQL_ATTR_APP_ROW_DESC` (ARD)
- `SQL_ATTR_IMP_PARAM_DESC` (IPD)
- `SQL_ATTR_IMP_ROW_DESC` (IRD)

For example, the following gives access to the statement’s implicitly allocated implementation parameter descriptor:

```

/* dbuse. c */
/* ... */
sqlrc = SQLGetStmtAttr ( hstmt,
                        SQL_ATTR_IMP_PARAM_DESC,
                        &hIPD,
                        SQL_IS_POINTER,
                        NULL);

```

Note: The descriptors whose handles are obtained in this manner will still be freed when the statement for which they were allocated is freed.

Explicitly allocated descriptors

An application can explicitly allocate application descriptors. It is not possible, however, to allocate implementation descriptors.

An application descriptor can be explicitly allocated any time the application is connected to the database. To explicitly allocate the application descriptor, call `SQLAllocHandle()` with a *HandleType* of `SQL_HANDLE_DESC`. For example, the following call explicitly allocates an application row descriptor:

```
rc = SQLAllocHandle( SQL_HANDLE_DESC, hdbc, &hARD );
```

To use an explicitly allocated application descriptor instead of a statement's implicitly allocated descriptor, call `SQLSetStmtAttr()`, and pass the statement handle, the descriptor handle, and an *Attribute* value of either:

- `SQL_ATTR_APP_PARAM_DESC` (APD), or
- `SQL_ATTR_APP_ROW_DESC` (ARD)

When there are explicitly and implicitly allocated descriptors, the explicitly specified one is used. An explicitly allocated descriptor can be associated with more than one statement.

Field initialization:

When an application row descriptor is allocated, its fields are initialized to the values listed in the descriptor header and record field initialization values documentation. The `SQL_DESC_TYPE` field is set to `SQL_DEFAULT` which provides for a standard treatment of database data for presentation to the application. The application may specify different treatment of the data by setting fields of the descriptor record.

The initial value of the `SQL_DESC_ARRAY_SIZE` header field is 1. To enable multirow fetch, the application can set this value in an ARD to the number of rows in a rowset.

There are no default values for the fields of an IRD. The fields are set when there is a prepared or executed statement.

The following fields of an IPD are undefined until a call to `SQLPrepare()` automatically populates them:

- `SQL_DESC_CASE_SENSITIVE`
- `SQL_DESC_FIXED_PREC_SCALE`
- `SQL_DESC_TYPE_NAME`
- `SQL_DESC_DESC_UNSIGNED`
- `SQL_DESC_LOCAL_TYPE_NAME`

Automatic population of the IPD:

There are times when the application will need to discover information about the parameters of a prepared SQL statement. A good example is when a dynamically generated query is prepared; the application will not know anything about the parameters in advance. If the application enables automatic population of the IPD, by setting the `SQL_ATTR_ENABLE_AUTO_IPD` statement attribute to `SQL_TRUE` (using `SQLSetStmtAttr()`), then the fields of the IPD are automatically populated to describe the parameter. This includes the data type, precision, scale, and so on (the same information that `SQLDescribeParam()` returns). The application can use this information to determine if data conversion is required, and which application buffer is the most appropriate to bind the parameter to.

Automatic population of the IPD involves some overhead. If it is not necessary for this information to be automatically gathered by the CLI driver then the `SQL_ATTR_ENABLE_AUTO_IPD` statement attribute should be set to `SQL_FALSE`.

When automatic population of the IPD is active, each call to `SQLPrepare()` causes the fields of the IPD to be updated. The resulting descriptor information can be retrieved by calling the following functions:

- `SQLGetDescField()`
- `SQLGetDescRec()`
- `SQLDescribeParam()`

Freeing of descriptors:

Explicitly allocated descriptors

When an explicitly allocated descriptor is freed, all statement handles to which the freed descriptor applied automatically revert to the original descriptors implicitly allocated for them.

Explicitly allocated descriptors can be freed in one of two ways:

- by calling `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_DESC`
- by freeing the connection handle that the descriptor is associated with

Implicitly allocated descriptors

An implicitly allocated descriptor can be freed in one of the following ways:

- by calling `SQLDisconnect()` which drops any statements or descriptors open on the connection
- by calling `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_STMT` to free the statement handle and all of the implicitly allocated descriptors associated with the statement

An implicitly allocated descriptor cannot be freed by calling `SQLFreeHandle()` with a *HandleType* of `SQL_HANDLE_DESC`.

Related concepts:

- “Descriptors in CLI applications” on page 151

Related reference:

- “Descriptor FieldIdentifier argument values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “Descriptor header and record field initialization values (CLI)” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFreeHandle function (CLI) - Free handle resources” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbuse.c -- How to use a database”

Descriptor manipulation with descriptor handles in CLI applications

Descriptors can be manipulated using descriptor handles or with DB2 CLI functions that do not use descriptor handles. This topic describes accessing descriptors through descriptor handles. The handle of an explicitly allocated descriptor is returned in the *OutputHandlePtr* argument when the application calls `SQLAllocHandle()` to allocate the descriptor. The handle of an implicitly allocated descriptor is obtained by calling `SQLGetStmtAttr()` with either `SQL_ATTR_IMP_PARAM_DESC` or `SQL_ATTR_IMP_ROW_DESC`.

Retrieval of descriptor field values:

The DB2 CLI function `SQLGetDescField()` can be used to obtain a single field of a descriptor record. `SQLGetDescRec()` retrieves the settings of multiple descriptor fields that affect the data type and storage of column or parameter data.

Setting of descriptor field values:

Two methods are available for setting descriptor fields: one field at a time or multiple fields at once.

Setting of individual fields:

Some fields of a descriptor are read-only, but others can be set using the function `SQLSetDescField()`. Refer to the list of header and record fields in the descriptor `FieldIdentifier` values documentation.

Record and header fields are set differently using `SQLSetDescField()` as follows:

Header fields

The call to `SQLSetDescField()` passes the header field to be set and a record number of 0. The record number is ignored since there is only one header field per descriptor. In this case the record number of 0 does not indicate the bookmark field.

Record fields

The call to `SQLSetDescField()` passes the record field to be set and a record number of 1 or higher, or 0 to indicate the bookmark field.

The application must follow the sequence of setting descriptor fields described in the `SQLSetDescField()` documentation when setting individual fields of a descriptor. Setting some fields will cause DB2 CLI to automatically set other fields. A consistency check will take place after the application follows the defined steps. This will ensure that the values in the descriptor fields are consistent.

If a function call that would set a descriptor fails, the content of the descriptor fields are undefined after the failed function call.

Setting of multiple fields:

A predefined set of descriptor fields can be set with one call rather than setting individual fields one at a time. `SQLSetDescRec()` sets the following fields for a single column or parameter:

- `SQL_DESC_TYPE`
- `SQL_DESC_OCTET_LENGTH`
- `SQL_DESC_PRECISION`
- `SQL_DESC_SCALE`
- `SQL_DESC_DATA_PTR`
- `SQL_DESC_OCTET_LENGTH_PTR`
- `SQL_DESC_INDICATOR_PTR`

(`SQL_DESC_DATETIME_INTERVAL_CODE` is also defined by ODBC but is not supported by DB2 CLI.)

For example, all of the descriptor fields listed above are set with the following call:

```
/* dbuse.c */
/* ... */
rc = SQLSetDescRec(hARD, 1, type, 0,
                  length, 0, 0, &id_no, &datalen, NULL);
```

Copying of descriptors:

One benefit of descriptors is the fact that a single descriptor can be used for multiple purposes. For instance, an ARD on one statement handle can be used as an APD on another statement handle.

There will be other instances, however, where the application will want to make a copy of the original descriptor, then modify certain fields. In this case `SQLCopyDesc()` is used to overwrite the fields of an existing descriptor with the values from another descriptor. Only fields that are defined for both the source and target descriptors are copied (with the exception of the `SQL_DESC_ALLOC_TYPE` field which cannot be changed).

Fields can be copied from any type of descriptor, but can only be copied to an application descriptor (APD or ARD) or an IPD. Fields cannot be copied to an IRD. The descriptor's allocation type will not be changed by the copy procedure (again, the `SQL_DESC_ALLOC_TYPE` field cannot be changed).

Related concepts:

- "Consistency checks for descriptors in CLI applications" on page 154
- "Descriptors in CLI applications" on page 151
- "Handles in CLI" on page 15

Related reference:

- "Descriptor FieldIdentifier argument values (CLI)" in *Call Level Interface Guide and Reference, Volume 2*
- "Descriptor header and record field initialization values (CLI)" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record" in *Call Level Interface Guide and Reference, Volume 2*

- “SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescField function (CLI) - Set a single field of a descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbuse.c -- How to use a database”

Descriptor manipulation without using descriptor handles in CLI applications

Many CLI functions make use of descriptors, but the application itself does not need to manipulate them directly. Instead, the application can use a different function which will set or retrieve one or more fields of a descriptor as well as perform other functions. This category of CLI functions is called *concise* functions. `SQLBindCol()` is an example of a concise function that manipulates descriptor fields.

In addition to manipulating multiple fields, concise functions are called without explicitly specifying the descriptor handle. The application does not even need to retrieve the descriptor handle to use a concise function.

The following types of concise functions exist:

- The functions `SQLBindCol()` and `SQLBindParameter()` bind a column or parameter by setting the descriptor fields that correspond to their arguments. These functions also perform other tasks unrelated to descriptors.

If required, an application can also use the descriptor calls directly to modify individual details of a binding. In this case the descriptor handle must be retrieved, and the functions `SQLSetDescField()` or `SQLSetDescRec()` are called to modify the binding.

- The following functions always retrieve values in descriptor fields:
 - `SQLColAttribute()`
 - `SQLDescribeCol()`
 - `SQLDescribeParam()`
 - `SQLNumParams()`
 - `SQLNumResultCols()`
- The functions `SQLSetDescRec()` and `SQLGetDescRec()` set or get the multiple descriptor fields that affect the data type and storage of column or parameter data. A single call to `SQLSetDescRec()` can be used to change the values used in the binding of a column or parameter.
- The functions `SQLSetStmtAttr()` and `SQLGetStmtAttr()` modify or return descriptor fields in some cases, depending on which statement attribute is specified. Refer to the “Values Stored in a Descriptor” section of the descriptors documentation for more information.

Related concepts:

- “Descriptors in CLI applications” on page 151
- “Handles in CLI” on page 15

Related reference:

- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescField function (CLI) - Set a single field of a descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*

Chapter 17. Environment, connection, and statement attributes

Environment, connection, and statement attributes in CLI applications

Environments, connections, and statements each have a defined set of attributes (or options). All attributes can be queried by the application, but only some attributes can be changed from their default values. By changing attribute values, the application can change the behavior of DB2 CLI.

An environment handle has attributes which affect the behavior of DB2 CLI functions under that environment. The application can specify the value of an attribute by calling `SQLSetEnvAttr()` and can obtain the current attribute value by calling `SQLGetEnvAttr()`. `SQLSetEnvAttr()` can only be called before any connection handles have been allocated for the environment handle. For details on each environment attribute, refer to the list of CLI environment attributes.

A connection handle has attributes which affect the behavior of DB2 CLI functions under that connection. Of the attributes that can be changed:

- Some can be set any time once the connection handle is allocated.
- Some can be set only before the actual connection has been established.
- Some can be set any time after the connection has been established.
- Some can be set after the connection has been established, but only while there are no outstanding transactions or open cursors.

The application can change the value of connection attributes by calling `SQLSetConnectAttr()` and can obtain the current value of an attribute by calling `SQLGetConnectAttr()`. An example of a connection attribute which can be set any time after a handle is allocated is the auto-commit option `SQL_ATTR_AUTOCOMMIT`. For details on each connection attribute, refer to the list of CLI connection attributes.

A statement handle has attributes which affect the behavior of CLI functions executed using that statement handle. Of the statement attributes that can be changed:

- Some attributes can be set, but currently are limited to only one specific value.
- Some attributes can be set any time after the statement handle has been allocated.
- Some attributes can only be set if there is no open cursor on that statement handle.

The application can specify the value of any statement attribute that can be set by calling `SQLSetStmtAttr()` and can obtain the current value of an attribute by calling `SQLGetStmtAttr()`. For details on each statement attribute, refer to the list of CLI statement attributes.

The `SQLSetConnectAttr()` function cannot be used to set statement attributes. This was supported in versions of DB2 CLI prior to version 5.

Many applications just use the default attribute settings; however, there may be situations where some of these defaults are not suitable for a particular user of the application. Some default values can be changed by setting the CLI/ODBC

configuration keywords. DB2 CLI provides end users with two methods of setting some configuration keywords. The first method is to specify the keyword and its new default attribute value(s) in the connection string input to the `SQLDriverConnect()` and `SQLBrowseConnect()` functions. The second method involves the specification of the new default attribute value(s) in a DB2 CLI initialization file using CLI/ODBC configuration keywords.

The DB2 CLI initialization file can be used to change default values for all DB2 CLI applications on that workstation. This may be the end user's only means of changing the defaults if the application does not provide a means for the user to provide default attribute values in the `SQLDriverConnect()` connection string. Default attribute values that are specified on `SQLDriverConnect()` override the values in the DB2 CLI initialization file for that particular connection.

The mechanisms for changing defaults are intended for end user tuning; application developers must use the appropriate set-attribute function. If an application does call a set-attribute or option function with a value different from the initialization file or the connection string specification, then the initial default value is overridden and the new value takes effect.

The diagram below shows the addition of the attribute functions to the basic connect scenario.

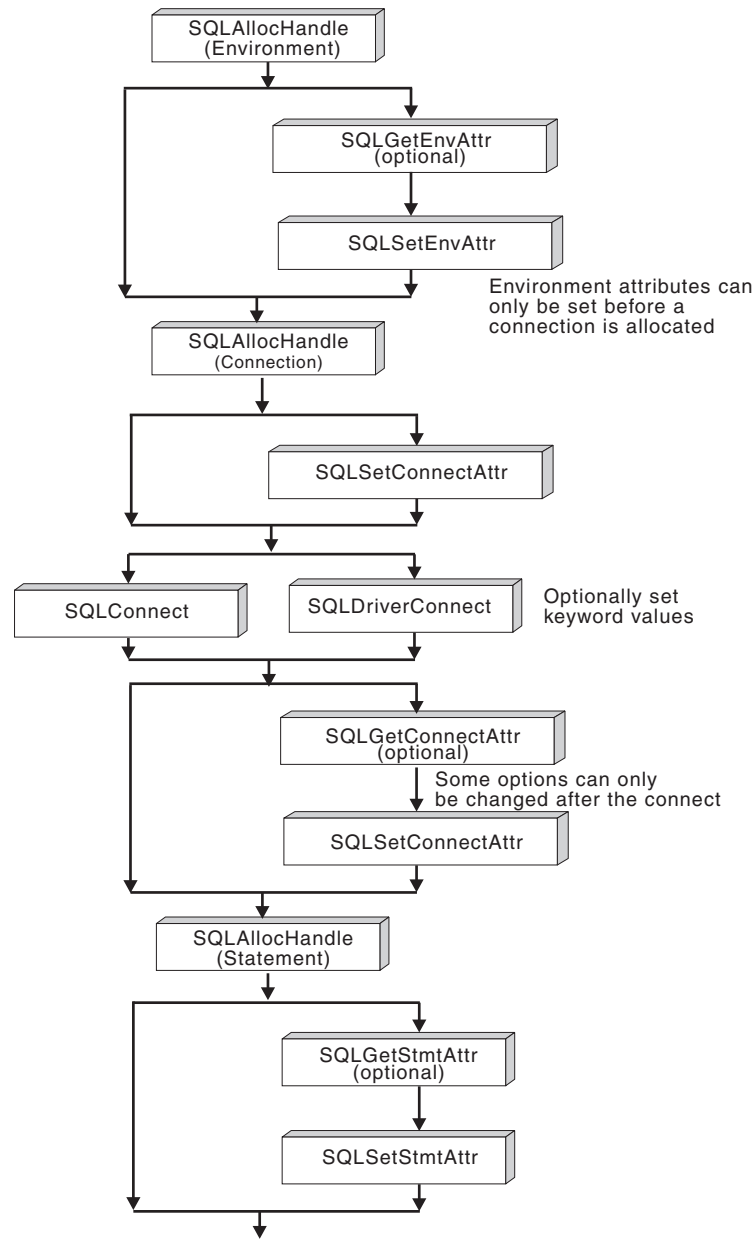


Figure 11. Setting and retrieving attributes (options)

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Handles in CLI” on page 15
- “Programming hints and tips for CLI applications” on page 55

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetConnectAttr function (CLI) - Get current attribute setting” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetConnectAttr function (CLI) - Set connection attributes” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetEnvAttr function (CLI) - Set environment attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

Related samples:

- “dbuse.c -- How to use a database”
- “spcall.c -- Call individual stored procedures”
- “tbread.c -- How to read data from tables”

Chapter 18. Querying system catalog information

Catalog functions for querying system catalog information in CLI applications

One of the first tasks an application often performs is to display a list of tables from which one or more are selected by the user. Although the application can issue its own queries against the database system catalog to get catalog information for such a DB2 command, it is best that the application calls the DB2 CLI catalog functions instead. These catalog functions, also called schema functions, provide a generic interface to issue queries and return consistent result sets across the DB2 family of servers. This allows the application to avoid server-specific and release-specific catalog queries.

The catalog functions operate by returning to the application a result set through a statement handle. Calling these functions is conceptually equivalent to using `SQLExecDirect()` to execute a select against the system catalog tables. After calling these functions, the application can fetch individual rows of the result set as it would process column data from an ordinary `SQLFetch()`. The DB2 CLI catalog functions are:

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLForeignKeys()`
- `SQLGetTypeInfo()`
- `SQLPrimaryKeys()`
- `SQLProcedureColumns()`
- `SQLProcedures()`
- `SQLSpecialColumns()`
- `SQLStatistics()`
- `SQLTablePrivileges()`
- `SQLTables()`

The result sets returned by these functions are defined in the descriptions for each catalog function. The columns are defined in a specified order. In future releases, other columns may be added to the end of each defined result set, therefore applications should be written in a way that would not be affected by such changes.

Some of the catalog functions result in execution of fairly complex queries, and for this reason should only be called when needed. It is recommended that the application save the information returned rather than making repeated calls to get the same information.

Related concepts:

- “Input arguments on catalog functions in CLI applications” on page 168

Related reference:

- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*

Input arguments on catalog functions in CLI applications

All of the catalog functions have *CatalogName* and *SchemaName* (and their associated lengths) on their input argument list. Other input arguments may also include *TableName*, *ProcedureName*, or *ColumnName* (and their associated lengths). These input arguments are used to either identify or constrain the amount of information to be returned.

Input arguments to catalog functions may be treated as ordinary arguments or pattern value arguments. An ordinary argument is treated as a literal, and the case of letters is significant. These arguments limit the scope of the query by identifying the object of interest. An error results if the application passes a null pointer for the argument.

Some catalog functions accept pattern values on some of their input arguments. For example, `SQLColumnPrivileges()` treats *SchemaName* and *TableName* as ordinary arguments and *ColumnName* as a pattern value. Refer to the "Function Arguments" section of the specific catalog function to see if a particular input argument accepts pattern values.

Inputs treated as pattern values are used to constrain the size of the result set by including only matching rows as though the underlying query's WHERE clause contained a LIKE predicate. If the application passes a null pointer for a pattern value input, the argument is not used to restrict the result set (that is, there is no corresponding LIKE in the WHERE clause). If a catalog function has more than one pattern value input argument, they are treated as though the LIKE predicates of the WHERE clauses in the underlying query were joined by AND; a row appears in this result set only if it meets all the conditions of the LIKE predicates.

Each pattern value argument can contain:

- The underscore (`_`) character which stands for any single character.
- The percent (`%`) character which stands for any sequence of zero or more characters. Note that providing a pattern value containing a single `%` is equivalent to passing a null pointer for that argument.
- Characters with no special meaning which stand for themselves. The case of a letter is significant.

These argument values are used on conceptual LIKE predicate(s) in the WHERE clause. To treat the metadata characters (`_`, `%`) as themselves, an escape character must immediately precede the `_` or `%`. The escape character itself can be specified as part of the pattern by including it twice in succession. An application can determine the escape character by calling `SQLGetInfo()` with `SQL_SEARCH_PATTERN_ESCAPE`.

For example, the following calls would retrieve all the tables that start with 'ST':

```
/* tbinfo.c */
/* ... */
struct
{
    SQLINTEGER ind ;
    SQLCHAR    val[129] ;
} tbQualifier, tbSchema, tbName, tbType;

struct
{
    SQLINTEGER ind ;
    SQLCHAR val[255] ;
} tbRemarks;
```

```

SQLCHAR tbSchemaPattern[] = "%";
SQLCHAR tbNamePattern[] = "ST%"; /* all the tables starting with ST */

/* ... */
sqlrc = SQLTables( hstmt, NULL, 0,
                  tbSchemaPattern, SQL_NTS,
                  tbNamePattern, SQL_NTS,
                  NULL, 0);

/* ... */

/* bind columns to variables */
sqlrc = SQLBindCol( hstmt, 1, SQL_C_CHAR, tbQualifier.val, 129,
                  &tbQualifier.ind );
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 2, SQL_C_CHAR, tbSchema.val, 129,
                  &tbSchema.ind );
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 3, SQL_C_CHAR, tbName.val, 129,
                  &tbName.ind );
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 4, SQL_C_CHAR, tbType.val, 129,
                  &tbType.ind );
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 5, SQL_C_CHAR, tbRemarks.val, 255,
                  &tbRemarks.ind );
STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
sqlrc = SQLFetch( hstmt );
/* ... */
while (sqlrc != SQL_NO_DATA_FOUND)
{ /* ... */
    sqlrc = SQLFetch( hstmt );
    /* ... */
}

```

Related concepts:

- “Catalog functions for querying system catalog information in CLI applications” on page 167

Related samples:

- “tbinfo.c -- How to get information about tables from the system catalog tables”

Chapter 19. Vendor escape clauses

Vendor escape clauses in CLI applications

The X/Open SQL CAE specification defined an **escape clause** as: “a syntactic mechanism for vendor-specific SQL extensions to be implemented in the framework of standardized SQL”. Both DB2 CLI and ODBC support vendor escape clauses as defined by X/Open.

Currently, escape clauses are used extensively by ODBC to define SQL extensions. DB2 CLI translates the ODBC extensions into the correct DB2 syntax. The `SQLNativeSql()` function can be used to display the resulting syntax.

If an application is only going to access DB2 data sources, then there is no reason to use the escape clauses. If an application is going to access other data sources that offer the same support through a different syntax, then the escape clauses increase the portability of the application.

DB2 CLI used both the standard and shorthand syntax for escape clauses. The standard syntax has been deprecated (although DB2 CLI still supports it). An escape clause using the standard syntax took the form:

```
--(*vendor(vendor-identifier),  
product(product-identifier) extended SQL text*)--
```

Applications should now only use the shorthand syntax, as described below, to remain current with the latest ODBC standards.

Shorthand escape clause syntax:

The format of an escape clause definition is:

```
{ extended SQL text }
```

to define the following SQL extensions:

- Extended date, time, timestamp data
- Outer join
- LIKE predicate
- Stored procedure call
- Extended scalar functions
 - Numeric functions
 - String functions
 - System functions

ODBC date, time, timestamp data:

The ODBC escape clauses for date, time, and timestamp data are:

```
{d 'value'}  
{t 'value'}  
{ts 'value'}
```

d indicates *value* is a date in the *yyyy-mm-dd* format,

t indicates *value* is a time in the *hh:mm:ss* format

ts indicates *value* is a timestamp in the *yyyy-mm-dd hh:mm:ss[.f...]* format.

For example, the following statement can be used to issue a query against the **EMPLOYEE** table:

```
SELECT * FROM EMPLOYEE WHERE HIREDATE={d '1994-03-29'}
```

DB2 CLI will translate the above statement to a DB2 format. `SQLNativeSql()` can be used to return the translated statement.

The ODBC escape clauses for date, time, and timestamp literals can be used in input parameters with a C data type of `SQL_C_CHAR`.

ODBC outer join:

The ODBC escape clause for outer join is:

```
{oj outer-join}
```

where *outer join* is

```
table-name {LEFT | RIGHT | FULL} OUTER JOIN  
{table-name | outer-join}  
ON search-condition
```

For example, DB2 CLI will translate the following statement:

```
SELECT * FROM {oj T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C3}  
WHERE T1.C2>20
```

to IBM's format, which corresponds to the SQL92 outer join syntax:

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C3 WHERE T1.C2>20
```

Note: Not all DB2 servers support outer join. To determine if the current server supports outer joins, call `SQLGetInfo()` with the `SQL_SQL92_RELATIONAL_JOIN_OPERATORS` and `SQL_OJ_CAPABILITIES` options.

LIKE predicate:

In a SQL LIKE predicate, the metacharacter `%` matches zero or more of any character, and the metacharacter `_` matches any one character. The SQL ESCAPE clause allows the definition of patterns intended to match values that contain the actual percent and underscore characters by preceding them with an escape character. The escape clause ODBC uses to define the LIKE predicate escape character is:

```
{escape 'escape-character'}
```

where *escape-character* is any character supported by the DB2 rules governing the use of the SQL ESCAPE clause.

As an example of how to use an "escape" ODBC escape clause, suppose you had a table `Customers` with the columns `Name` and `Growth`. The `Growth` column contains data having the metacharacter `'%'`. The following statement would select all of the values from `Name` that have values in `Growth` only between 10% and 19%, excluding 100% and above:

```
SELECT Name FROM Customers WHERE Growth LIKE '1_\\%' {escape '\\'}
```

Applications that are not concerned about portability across different vendor DBMS products should pass an SQL ESCAPE clause directly to the data source. To determine when LIKE predicate escape characters are supported by a particular

DB2 data source, an application should call `SQLGetInfo()` with the `SQL_LIKE_ESCAPE_CLAUSE` information type.

Stored procedure call:

The ODBC escape clause for calling a stored procedure is:

```
{[?=call procedure-name[[parameter][,parameter]...]}
```

where:

- `[?=call]` indicates the optional parameter marker for the return value
- `procedure-name` specifies the name of a procedure stored at the data source
- `parameter` specifies a procedure parameter.

A procedure may have zero or more parameters.

ODBC specifies the optional parameter `?=` to represent the procedure's return value, which if present, will be stored in the location specified by the first parameter marker as defined via `SQLBindParameter()`. DB2 CLI will return the return code as the procedure's return value if `?=` is present in the escape clause. If `?=` is not present, and if the stored procedure return code is not `SQL_SUCCESS`, then the application can retrieve diagnostics, including the `SQLCODE`, using the `SQLGetDiagRec()` and `SQLGetDiagField()` functions. DB2 CLI supports literals as procedure arguments, however vendor escape clauses must be used. For example, the following statement would not succeed: `CALL storedproc ('aaaa', 1)`, but this statement would: `{CALL storedproc ('aaaa', 1)}`. If a parameter is an output parameter, it must be a parameter marker.

For example, DB2 CLI will translate the following statement:

```
{CALL NETB94(?, ?, ?)}
```

To an internal CALL statement format:

```
CALL NEBT94(?, ?, ?)
```

ODBC scalar functions:

Scalar functions such as string length, substring, or trim can be used on columns of a result set and on columns that restrict rows of a result set. The ODBC escape clause for scalar functions is:

```
{fn scalar-function}
```

Where, *scalar-function* can be any function listed in the list of extended scalar functions.

For example, DB2 CLI will translate the following statement:

```
SELECT {fn CONCAT(FIRSTNAME, LASTNAME)} FROM EMPLOYEE
```

to:

```
SELECT FIRSTNAME CONCAT LASTNAME FROM EMPLOYEE
```

`SQLNativeSql()` can be called to obtain the translated SQL statement.

To determine which scalar functions are supported by the current server referenced by a specific connection handle, call `SQLGetInfo()` with the options:

SQL_NUMERIC_FUNCTIONS, SQL_STRING_FUNCTIONS,
SQL_SYSTEM_FUNCTIONS, and SQL_TIMEDATE_FUNCTIONS.

Related reference:

- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetInfo function (CLI) - Get general information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLNativeSql function (CLI) - Get native SQL text” in *Call Level Interface Guide and Reference, Volume 2*
- “CALL statement” in *SQL Reference, Volume 2*
- “Extended scalar functions for CLI applications” on page 174
- “LIKE predicate” in *SQL Reference, Volume 1*
- “SELECT statement” in *SQL Reference, Volume 2*

Related samples:

- “dbnative.c -- How to translate a statement that contains an ODBC escape clause”

Extended scalar functions for CLI applications

The following functions are defined by ODBC using vendor escape clauses. Each function can be called using the escape clause syntax, or calling the equivalent DB2 function.

These functions are presented in the following categories:

- “String functions” on page 175
- “Numeric functions” on page 176
- “Date and time functions” on page 178
- “System functions” on page 181
- “Conversion function” on page 181

The tables in the following sections indicates for which servers (and the earliest versions) that the function can be accessed, when called from an application using DB2 CLI.

All errors detected by the following functions, when connected to a DB2 Version 5 or later server, will return SQLSTATE 38552. The text portion of the message is of the form SYSFUN:*nn* where *nn* is one of the following reason codes:

- 01 Numeric value out of range
- 02 Division by zero
- 03 Arithmetic overflow or underflow
- 04 Invalid date format
- 05 Invalid time format
- 06 Invalid timestamp format
- 07 Invalid character representation of a timestamp duration
- 08 Invalid interval type (must be one of 1, 2, 4, 8, 16, 32, 64, 128, 256)
- 09 String too long
- 10 Length or position in string function out of range
- 11 Invalid character representation of a floating point number

String functions:

The string functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Character string literals used as arguments to scalar functions must be bounded by single quotes.
- Arguments denoted as *string_exp* can be the name of a column, a string literal, or the result of another scalar function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, or SQL_CLOB.
- Arguments denoted as *start*, *length*, *code* or *count* can be a numeric literal or the result of another scalar function, where the underlying data type is integer based (SQL_SMALLINT, SQL_INTEGER).
- The first character in the string is considered to be at position 1.

Table 13. String scalar functions

String scalar function	Description	Servers that support the function
ASCII(<i>string_exp</i>)	Returns the ASCII code value of the leftmost character of <i>string_exp</i> as an integer.	DB2 for workstation
CHAR(<i>code</i>)	Returns the character that has the ASCII code value specified by <i>code</i> . The value of <i>code</i> should be between 0 and 255; otherwise, the return value is null.	DB2 for workstation
CONCAT(<i>string_exp1</i> , <i>string_exp2</i>)	Returns a character string that is the result of concatenating <i>string_exp2</i> to <i>string_exp1</i> .	DB2 for workstation, MVS™, VM/VSE, AS/400
DIFFERENCE(<i>string_exp1</i> , <i>string_exp2</i>)	Returns an integer value indicating the difference between the values returned by the SOUNDEX function for <i>string_exp1</i> and <i>string_exp2</i> .	DB2 for workstation
INSERT(<i>string_exp1</i> , <i>start</i> , <i>length</i> , <i>string_exp2</i>)	Returns a character string where <i>length</i> number of characters beginning at <i>start</i> has been replaced by <i>string_exp2</i> which contains <i>length</i> characters.	DB2 for workstation, MVS, VM/VSE, AS/400
LCASE(<i>string_exp</i>)	Converts all uppercase characters in <i>string_exp</i> to lowercase.	DB2 for workstation, VM/VSE
LEFT(<i>string_exp</i> , <i>count</i>)	Returns the leftmost <i>count</i> of characters of <i>string_exp</i> .	DB2 for workstation, MVS, VM/VSE, AS/400
LENGTH(<i>string_exp</i>)	Returns the number of characters in <i>string_exp</i> , excluding trailing blanks and the string termination character. Note: Trailing blanks are included for DB2 for MVS/ESA™.	DB2 for workstation, MVS, VM/VSE, AS/400
LOCATE(<i>string_exp1</i> , <i>string_exp2</i> [<i>start</i>])	Returns the starting position of the first occurrence of <i>string_exp1</i> within <i>string_exp2</i> . The search for the first occurrence of <i>string_exp1</i> begins with first character position in <i>string_exp2</i> unless the optional argument, <i>start</i> , is specified. If <i>start</i> is specified, the search begins with the character position indicated by the value of <i>start</i> . The first character position in <i>string_exp2</i> is indicated by the value 1. If <i>string_exp1</i> is not found within <i>string_exp2</i> , the value 0 is returned.	DB2 for workstation

Table 13. String scalar functions (continued)

String scalar function	Description	Servers that support the function
<i>LTRIM(string_exp)</i>	Returns the characters of <i>string_exp</i> with the leading blanks removed.	DB2 for workstation, AS/400
<i>REPEAT(string_exp, count)</i>	Returns a character string composed of <i>string_exp</i> repeated <i>count</i> times.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>REPLACE(string_exp1, string_exp2, string_exp3)</i>	Replaces all occurrences of <i>string_exp2</i> in <i>string_exp1</i> with <i>string_exp3</i> .	DB2 for workstation
<i>RIGHT(string_exp, count)</i>	Returns the rightmost count of characters of <i>string_exp</i> .	DB2 for workstation, MVS, VM/VSE, AS/400
<i>RTRIM(string_exp)</i>	Returns the characters of <i>string_exp</i> with trailing blanks removed.	DB2 for workstation, VM/VSE, AS/400
<i>SOUNDEX(string_exp1)</i>	Returns a four character code representing the sound of <i>string_exp1</i> . Note that different data sources use different algorithms to represent the sound of <i>string_exp1</i> .	DB2 for workstation
<i>SPACE(count)</i>	Returns a character string consisting of <i>count</i> spaces.	DB2 for workstation
<i>SUBSTRING(string_exp, start, length)</i>	Returns a character string that is derived from <i>string_exp</i> beginning at the character position specified by <i>start</i> for <i>length</i> characters.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>UCASE(string_exp)</i>	Converts all lowercase characters in <i>string_exp</i> to uppercase.	DB2 for workstation, VM/VSE, AS/400

Numeric functions:

The numeric functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Arguments denoted as *numeric_exp* can be the name of a column, the result of another scalar function, or a numeric literal, where the underlying data type can be either floating point based (SQL_NUMERIC, SQL_DECIMAL, SQL_FLOAT, SQL_REAL, SQL_DOUBLE) or integer based (SQL_SMALLINT, SQL_INTEGER).
- Arguments denoted as *double_exp* can be the name of a column, the result of another scalar functions, or a numeric literal where the underlying data type is floating point based.
- Arguments denoted as *integer_exp* can be the name of a column, the result of another scalar functions, or a numeric literal, where the underlying data type is integer based.

Table 14. Numeric scalar functions

Numeric scalar function	Description	Servers that support the function
<i>ABS(numeric_exp)</i>	Returns the absolute value of <i>numeric_exp</i>	DB2 for workstation, AS/400

Table 14. Numeric scalar functions (continued)

Numeric scalar function	Description	Servers that support the function
ACOS(<i>double_exp</i>)	Returns the arccosine of <i>double_exp</i> as an angle, expressed in radians.	DB2 for workstation, AS/400
ASIN(<i>double_exp</i>)	Returns the arcsine of <i>double_exp</i> as an angle, expressed in radians.	DB2 for workstation, AS/400
ATAN(<i>double_exp</i>)	Returns the arctangent of <i>double_exp</i> as an angle, expressed in radians.	DB2 for workstation, AS/400
ATAN2(<i>double_exp1</i> , <i>double_exp2</i>)	Returns the arctangent of <i>x</i> and <i>y</i> coordinates specified by <i>double_exp1</i> and <i>double_exp2</i> , respectively, as an angle expressed in radians.	DB2 for workstation
CEILING(<i>numeric_exp</i>)	Returns the smallest integer greater than or equal to <i>numeric_exp</i> .	DB2 for workstation
COS(<i>double_exp</i>)	Returns the cosine of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for workstation, AS/400
COT(<i>double_exp</i>)	Returns the cotangent of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for workstation, AS/400
DEGREES(<i>numeric_exp</i>)	Returns the number of degrees converted from <i>numeric_exp</i> radians.	DB2 for workstation, AS/400 (3.6)
EXP(<i>double_exp</i>)	Returns the exponential value of <i>double_exp</i> .	DB2 for workstation, AS/400
FLOOR(<i>numeric_exp</i>)	Returns the largest integer less than or equal to <i>numeric_exp</i> .	DB2 for workstation, AS/400 (3.6)
LOG(<i>double_exp</i>)	Returns the natural logarithm of <i>double_exp</i> .	DB2 for workstation, AS/400
LOG10(<i>double_exp</i>)	Returns the base 10 logarithm of <i>double_exp</i> .	DB2 for workstation, AS/400
MOD(<i>integer_exp1</i> , <i>integer_exp2</i>)	Returns the remainder (modulus) of <i>integer_exp1</i> divided by <i>integer_exp2</i> .	DB2 for workstation, AS/400
PI()	Returns the constant value of pi as a floating point value.	DB2 for workstation, AS/400
POWER(<i>numeric_exp</i> , <i>integer_exp</i>)	Returns the value of <i>numeric_exp</i> to the power of <i>integer_exp</i> .	DB2 for workstation, AS/400 (3.6)
RADIANS(<i>numeric_exp</i>)	Returns the number of radians converted from <i>numeric_exp</i> degrees.	DB2 for workstation
RAND([<i>integer_exp</i>])	Returns a random floating point value using <i>integer_exp</i> as the optional seed value.	DB2 for workstation

Table 14. Numeric scalar functions (continued)

Numeric scalar function	Description	Servers that support the function
<i>ROUND(numeric_exp, integer_exp.)</i>	Returns <i>numeric_exp</i> rounded to <i>integer_exp</i> places right of the decimal point. If <i>integer_exp</i> is negative, <i>numeric_exp</i> is rounded to <i>integer_exp</i> places to the left of the decimal point.	DB2 for workstation
<i>SIGN(numeric_exp)</i>	Returns an indicator or the sign of <i>numeric_exp</i> . If <i>numeric_exp</i> is less than zero, -1 is returned. If <i>numeric_exp</i> equals zero, 0 is returned. If <i>numeric_exp</i> is greater than zero, 1 is returned.	DB2 for workstation
<i>SIN(double_exp)</i>	Returns the sine of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for workstation, AS/400
<i>SQRT(double_exp)</i>	Returns the square root of <i>double_exp</i> .	DB2 for workstation, AS/400
<i>TAN(double_exp)</i>	Returns the tangent of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for workstation, AS/400
<i>TRUNCATE(numeric_exp, integer_exp)</i>	Returns <i>numeric_exp</i> truncated to <i>integer_exp</i> places right of the decimal point. If <i>integer_exp</i> is negative, <i>numeric_exp</i> is truncated to <i>integer_exp</i> places to the left of the decimal point.	DB2 for workstation

Date and time functions:

The date and time functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Arguments denoted as *timestamp_exp* can be the name of a column, the result of another scalar function, or a time, date, or timestamp literal.
- Arguments denoted as *date_exp* can be the name of a column, the result of another scalar function, or a date or timestamp literal, where the underlying data type can be character based, or date or timestamp based.
- Arguments denoted as *time_exp* can be the name of a column, the result of another scalar function, or a time or timestamp literal, where the underlying data types can be character based, or time or timestamp based.

Table 15. Date and time scalar functions

Date and time scalar function	Description	Servers that support the function
<i>CURDATE()</i>	Returns the current date as a date value.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>CURTIME()</i>	Returns the current local time as a time value.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>DAYNAME(date_exp)</i>	Returns a character string containing the name of the day (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday) for the day portion of <i>date_exp</i> .	DB2 for workstation

Table 15. Date and time scalar functions (continued)

Date and time scalar function	Description	Servers that support the function
<i>DAYOFMONTH</i> (<i>date_exp</i>)	Returns the day of the month in <i>date_exp</i> as an integer value in the range of 1-31.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>DAYOFWEEK</i> (<i>date_exp</i>)	Returns the day of the week in <i>date_exp</i> as an integer value in the range 1-7, where 1 represents Sunday.	DB2 for workstation, AS/400 (3.6)
<i>DAYOFWEEK_ISO</i> (<i>date_exp</i>)	Returns the day of the week in <i>date_exp</i> as an integer value in the range 1-7, where 1 represents Monday. Note the difference between this function and the <i>DAYOFWEEK</i> () function, where 1 represents Sunday.	DB2 for workstation, AS/400 (3.6)
<i>DAYOFYEAR</i> (<i>date_exp</i>)	Returns the day of the year in <i>date_exp</i> as an integer value in the range 1-366.	DB2 for workstation, AS/400 (3.6)
<i>HOUR</i> (<i>time_exp</i>)	Returns the hour in <i>time_exp</i> as an integer value in the range of 0-23.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>JULIAN_DAY</i> (<i>date_exp</i>)	Returns the number of days between <i>date_exp</i> and January 1, 4712 B.C. (the start of the Julian date calendar).	DB2 for workstation
<i>MINUTE</i> (<i>time_exp</i>)	Returns the minute in <i>time_exp</i> as integer value in the range of 0-59.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>MONTH</i> (<i>date_exp</i>)	Returns the month in <i>date_exp</i> as an integer value in the range of 1-12.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>MONTHNAME</i> (<i>date_exp</i>)	Returns a character string containing the name of month (January, February, March, April, May, June, July, August, September, October, November, December) for the month portion of <i>date_exp</i> .	DB2 for workstation
<i>NOW</i> ()	Returns the current date and time as a timestamp value.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>QUARTER</i> (<i>date_exp</i>)	Returns the quarter in <i>date_exp</i> as an integer value in the range of 1-4.	DB2 for workstation, AS/400 (3.6)
<i>SECOND</i> (<i>time_exp</i>)	Returns the second in <i>time_exp</i> as an integer value in the range of 0-59.	DB2 for workstation, MVS, VM/VSE, AS/400
<i>SECONDS_SINCE_MIDNIGHT</i> (<i>time_exp</i>)	Returns the number of seconds in <i>time_exp</i> relative to midnight as an integer value in the range of 0-86400. If <i>time_exp</i> includes a fractional seconds component, the fractional seconds component will be discarded.	DB2 for workstation

Table 15. Date and time scalar functions (continued)

Date and time scalar function	Description	Servers that support the function
<p><i>TIMESTAMPADD(interval, integer_exp, timestamp_exp)</i></p>	<p>Returns the timestamp calculated by adding <i>integer_exp</i> intervals of type <i>interval</i> to <i>timestamp_exp</i>. Valid values of interval are:</p> <ul style="list-style-type: none"> • SQL_TSI_FRAC_SECOND • SQL_TSI_SECOND • SQL_TSI_MINUTE • SQL_TSI_HOUR • SQL_TSI_DAY • SQL_TSI_WEEK • SQL_TSI_MONTH • SQL_TSI_QUARTER • SQL_TSI_YEAR <p>where fractional seconds are expressed in billionths of a second. If <i>timestamp_exp</i> specifies a time value and <i>interval</i> specifies days, weeks, months, quarters, or years, the date portion of <i>timestamp_exp</i> is set to the current date before calculating the resulting timestamp. If <i>timestamp_exp</i> is a date value and <i>interval</i> specifies fractional seconds, seconds, minutes, or hours, the time portion of <i>timestamp_exp</i> is set to 00:00:00.000000 before calculating the resulting timestamp. An application determines which intervals are supported by calling <i>SQLGetInfo()</i> with the SQL_TIMEDATE_ADD_INTERVALS option.</p>	<p>DB2 for workstation</p>
<p><i>TIMESTAMPDIFF(interval, timestamp_exp1, timestamp_exp2)</i></p>	<p>Returns the integer number of intervals of type <i>interval</i> by which <i>timestamp_exp2</i> is greater than <i>timestamp_exp1</i>. Valid values of interval are:</p> <ul style="list-style-type: none"> • SQL_TSI_FRAC_SECOND • SQL_TSI_SECOND • SQL_TSI_MINUTE • SQL_TSI_HOUR • SQL_TSI_DAY • SQL_TSI_WEEK • SQL_TSI_MONTH • SQL_TSI_QUARTER • SQL_TSI_YEAR <p>where fractional seconds are expressed in billionths of a second. If either timestamp expression is a time value and <i>interval</i> specifies days, weeks, months, quarters, or years, the date portion of that timestamp is set to the current date before calculating the difference between the timestamps. If either timestamp expression is a date value and <i>interval</i> specifies fractional seconds, seconds, minutes, or hours, the time portion of that timestamp is set to 0 before calculating the difference between the timestamps. An application determines which intervals are supported by calling <i>SQLGetInfo()</i> with the SQL_TIMEDATE_DIFF_INTERVALS option.</p>	<p>DB2 for workstation</p>
<p><i>WEEK(date_exp)</i></p>	<p>Returns the week of the year in <i>date_exp</i> as an integer value in the range of 1-54.</p>	<p>DB2 for workstation, AS/400 (3.6)</p>

Table 15. Date and time scalar functions (continued)

Date and time scalar function	Description	Servers that support the function
<i>WEEK_ISO(date_exp)</i>	Returns the week of the year in <i>date_exp</i> as an integer value in the range of 1-53. Week 1 is defined as the first week of the year to contain a Thursday. Therefore, Week1 is equivalent to the first week that contains Jan 4, since Monday is considered to be the first day of the week. Note that WEEK_ISO() differs from the current definition of WEEK(), which returns a value up to 54. For the WEEK() function, Week 1 is the week containing the first Saturday. This is equivalent to the week containing Jan. 1, even if the week contains only one day.	DB2 for workstation
<i>YEAR(date_exp)</i>	Returns the year in <i>date_exp</i> as an integer value in the range of 1-9999.	DB2 for workstation, MVS, VM/VSE, AS/400

For those functions that return a character string containing the name of the day of week or the name of the month, these character strings will be National Language Support enabled.

DAYOFWEEK_ISO() and WEEK_ISO() are automatically available in a database created in DB2 Version 7 or later. If a database was created prior to Version 7, these functions might not be available. To make DAYOFWEEK_ISO() and WEEK_ISO() functions available in such a database, use the **db2updb** system command.

System functions:

The system functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Arguments denoted as *exp* can be the name of a column, the result of another scalar function, or a literal.
- Arguments denoted as *value* can be a literal constant.

Table 16. System scalar functions

System scalar function	Description	Servers that support the function
<i>DATABASE()</i>	Returns the name of the database corresponding to the connection handle (<i>hdbc</i>). (The name of the database is also available via <i>SQLGetInfo()</i> by specifying the information type SQL_DATABASE_NAME.)	DB2 for workstation, MVS, VM/VSE, AS/400
<i>IFNULL(exp, value)</i>	If <i>exp</i> is null, <i>value</i> is returned. If <i>exp</i> is not null, <i>exp</i> is returned. The possible data type(s) of <i>value</i> must be compatible with the data type of <i>exp</i> .	DB2 for workstation, MVS, VM/VSE, AS/400
<i>USER()</i>	Returns the user's authorization name. (The user's authorization name is also available via <i>SQLGetInfo()</i> by specifying the information type SQL_USER_NAME.)	DB2 for workstation, MVS, VM/VSE, AS/400

Conversion function:

The conversion function is supported by DB2 CLI and defined by ODBC using vendor escape clauses.

Each driver and data source determines which conversions are valid between the possible data types. As the driver translates the ODBC syntax into native syntax it will reject the conversions that are not supported by the data source, even if the ODBC syntax is valid.

Use the function *SQLGetInfo()* with the appropriate convert function masks to determine which conversions are supported by the data source.

Table 17. Conversion Function

Conversion scalar function	Description	Servers that support the function
<i>CONVERT(expr_value, data_type)</i>	<ul style="list-style-type: none"> • <i>data_type</i> indicates the data type of the converted representation of <i>expr_value</i>, and can be either <code>SQL_CHAR</code> or <code>SQL_DOUBLE</code>. • <i>expr_value</i> is the value to convert. It can be of various types, depending on the conversions supported by the driver and data source. Use the function <i>SQLGetInfo()</i> with the appropriate convert function masks to determine which conversions are supported by the data source. 	DB2 for workstation

Related concepts:

- “Vendor escape clauses in CLI applications” on page 171

Related reference:

- “SQL symbolic and default data types for CLI applications” on page 41

Chapter 20. Mixing embedded SQL and DB2 CLI

Considerations for mixing embedded SQL and DB2 CLI

It is possible, and sometimes desirable, to use DB2 CLI in conjunction with embedded static SQL in an application. Consider the scenario where the application developer wishes to take advantage of the ease of use provided by the DB2 CLI catalog functions and maximize the portion of the application's processing where performance is critical. In order to mix the use of DB2 CLI and embedded SQL, the application must comply with the following rules:

- All connection management and transaction management must be performed completely using either DB2 CLI or embedded SQL - never a mixture of the two. Two options are available to the application:
 - it performs all connects and commits/rollbacks using DB2 CLI calls, and then calls functions written using embedded SQL;
 - or it performs all connects and commits/rollbacks using embedded SQL, and then calls functions that use DB2 CLI APIs, notably, a null connection.
- Query statement processing cannot straddle DB2 CLI and embedded SQL interfaces for the same statement. For example, the application cannot open a cursor using embedded SQL, and then call the DB2 CLI `SQLFetch()` function to retrieve row data.

Since DB2 CLI permits multiple connections, the `SQLSetConnection()` function must be called prior to executing any embedded SQL. This allows the application to explicitly specify the connection under which the embedded SQL processing is performed.

If the DB2 CLI application is multithreaded and also makes embedded SQL calls or DB2 API calls, then each thread must have a DB2 context.

Related concepts:

- “DB2 Call Level Interface (CLI) versus embedded SQL” on page 4

Related reference:

- “SQLSetConnection function (CLI) - Set connection handle” in *Call Level Interface Guide and Reference, Volume 2*

Related samples:

- “dbmconx.c -- How to use multiple databases with embedded SQL.”
- “dbusemx.sqc -- How to execute embedded SQL statements in CLI”

Chapter 21. CLI/ODBC/JDBC Static Profiling

Creating static SQL with CLI/ODBC/JDBC Static Profiling

The CLI/ODBC/JDBC Static Profiling feature enables an application's end users to replace the use of dynamic SQL with static SQL, potentially resulting in runtime performance improvement and better security from the package-based authorization mechanism.

Restrictions:

- When executing an application with pre-bound static SQL statements, dynamic registers that control the dynamic statement behavior will have no effect on the statements that are converted to static.
- If an application issues DDL (data definition language) statements for objects that are referenced in subsequent DML (data manipulation language) statements, you will find all of these statements in the capture file. The CLI/ODBC/JDBC Static Profiling Bind Tool, db2cap, will attempt to bind them. The bind attempt will be successful with DBMSs that support the VALIDATE(RUN) bind option, but it will fail with ones that do not. In this case, the application should not use Static Profiling.
- The database administrator (DBA) may edit the capture file to add, change, or remove SQL statements, based on application-specific requirements.

Before running the application during the profiling session, ensure that the following conditions have been noted:

- An SQL statement must have successfully executed (generated a positive SQLCODE) for it to be captured in a profiling session. In a statement matching session, unmatched dynamic statements will continue to execute as dynamic CLI/ODBC/JDBC calls.
- An SQL statement must be identical character-by-character to the one that was captured and bound to be a valid candidate for statement matching. Spaces are significant: for example, "COL = 1" is considered different than "COL=1". Use parameter markers in place of literals to improve match hits.

Be aware that there are times when not all dynamic CLI/ODBC calls can be captured and grouped into a static package. Possible reasons are:

- The application does not regularly free environment handles. During a capture session, statements captured under a particular environment handle are only written to the capture file or files when that environment handle is freed.
- The application has complex control flows that make it difficult to cover all runtime conditions in a single application run.
- The application executes SET statements to change register variables. These statements are not recorded. Note that there is a limited capability in match mode to detect dynamic SET SQLID and SET SCHEMA statements, and suspend executing static statements accordingly. However, for other SET statements, subsequent SQL statements which depend on the register variables being set may not behave properly.

- The application issues DML (Data Manipulation Language) statements. Depending on application complexities and the nature of these statements, either: (1) they may not be matched, or (2) they may not execute properly at runtime.

Since dynamic and static SQL are quite different, the DBA should always verify the behaviour of the application in static match mode before making it available to end users. Furthermore, while static SQL may offer improved runtime performance over dynamic SQL, this is not necessarily true for all statements. If testing shows that static execution decreases performance for a particular statement, the DBA can force that statement to be dynamically executed by removing the statement from the capture file. In addition, static SQL, unlike dynamic SQL, may require occasional rebinding of packages to maintain performance, particularly if the database objects referred to in the packages frequently change. If CLI/ODBC/JDBC Static Profiling does not fit the type of application you are running, there are other programming methods which allow you to obtain the benefits of static SQL, such as embedded SQL and stored procedures.

Procedure:

To create static SQL statements from existing dynamic SQL statements, perform the following steps:

1. Profile the application by capturing all the dynamic SQL statements issued by the application. This process is known as running the application in static capture mode. To turn on static capture mode, set the following CLI/ODBC configuration keywords for the CLI/ODBC/JDBC data source in the `db2cli.ini` configuration file, before running the application:
 - `StaticMode = CAPTURE`
 - `StaticPackage = qualified package name`
 - `StaticCapFile = capture file name`

For example:

```
[DSN1]
StaticMode = CAPTURE
StaticPackage = MySchema.MyPkg
StaticCapFile = E:\Shared\MyApp.cpt
```

Attention: For the `StaticPackage` keyword, ensure that you specify a schema name (MySchema in the sample above). If a schema is not specified, the name you provide will be considered to be the container name instead of the package name, and the package name will be blank.

The resulting static profile takes the form of a text-based *capture file*, containing information about the SQL statements captured.

The above example file yields the following results: Data Source Name 1 (DSN1) is set to capture mode; the package will be named MySchema.MyPkg; and the capture file, MyApp.cpt, will be saved in the E:\Shared\ directory. Until the `StaticMode` keyword is changed to a value other than CAPTURE, such as DISABLED which is used to turn off static capture mode, each subsequent run of this application will capture SQL statements and append them to the capture file MyApp.cpt. Only unique SQL statements will be captured however, as duplicate executions are ignored.

2. Optional: Set the CLI/ODBC configuration keyword `StaticLogFile` to generate a CLI/ODBC/JDBC Static Profiling log file. It contains useful information to determine the state of the statement capturing process.
3. Run the application.

Unique SQL statements will now be captured in the capture file. Duplicate statements are ignored.

4. Disable static capture mode by setting the CLI/ODBC configuration keyword `StaticMode` to `DISABLED`, or remove the keywords set in the first step from the `db2cli.ini` file.
5. Issue the `db2cap` command from the Command Line Processor. The `db2cap` utility will generate a static package based on the capture file. If the `db2cap` utility does not return a message indicating successful completion, then a statement in the capture file could not be statically bound. The DBA should remove the failing statement from the capture file and run the `db2cap` utility again.
6. Distribute a copy of the capture file, processed with `db2cap` to each end user of the application. If all users reside on the same client platform, an alternative is to place a read-only copy of this capture file in a network directory accessible to all users.
7. Enable your application for dynamic-to-static SQL statement mapping, known as static match mode. Do this by setting the following CLI/ODBC configuration keywords:
 - `StaticMode = MATCH`
 - `StaticCapFile = capture file name`

For example:

```
[DSN1]
StaticMode = MATCH
StaticCapFile = E:\Shared\MyApp.cpt
```

8. Optional: Set the CLI/ODBC configuration keyword `StaticLogFile` keyword to log useful information such as how many statements were matched (therefore statically executed) and how many statements were unmatched (therefore dynamically executed) during a match session. The DBA should use this information to verify that static profiling in match mode is yielding an acceptable match ratio before making static profiling available to end users.
9. Run the application.

Related concepts:

- “Capture file for CLI/ODBC/JDBC Static Profiling” on page 187
- “Characteristics and reasons for using static SQL” in *SQL Guide*

Related reference:

- “`db2cap` - CLI/ODBC static package binding tool command” in *Command Reference*
- “`StaticCapFile` CLI/ODBC configuration keyword” on page 368
- “`StaticLogFile` CLI/ODBC configuration keyword” on page 369
- “`StaticMode` CLI/ODBC configuration keyword” on page 370
- “`StaticPackage` CLI/ODBC configuration keyword” on page 370

Capture file for CLI/ODBC/JDBC Static Profiling

The capture file generated during static profiling is a text file. It contains the text of SQL statements and other associated information obtained in static capture mode. As well, it keeps track of a number of configurable bind options; some already contain specific values obtained from the capture run, and some are left blank, in which case the precompiler will use default values during package binding. Before

binding the package(s), the DBA may want to examine the capture file and make necessary changes to these bind options using a text editor.

To help you understand how to edit SQL statements, here is the description of the fields in a statement:

Field	Description
SQLID	If present, indicates the SCHEMA or SQLID when the statement was captured is different from the default QUALIFIER of the package(s).
SECTNO	Section number of the static package that the statement was bound to.
ISOLATION	Isolation level for the statement. It determines which one of the five possible package the statement belongs to.
STMTTEXT	Statement string
STMTTYPE	There are 3 possible values: <ul style="list-style-type: none"> • SELECT_CURSOR_WITHHOLD: SELECT statement using a withhold cursor • SELECT_CURSOR_NOHOLD: SELECT statement using a nohold cursor • OTHER: non-SELECT statements
CURSOR	Cursor name declared for the SELECT statement
INVARnn	Description of the n-th input variable The 7 comma-separated fields refer to: <ol style="list-style-type: none"> 1. SQL data type 2. Length of the data. For decimal or floating point types, this is the precision. 3. For decimal or floating point types only, this is the scale. 4. TRUE if the character data is a for-bit-data type; otherwise FALSE. 5. TRUE if the variable is nullable; otherwise FALSE. 6. Column name 7. SQL_NAMED if this variable refers to a real column name; SQL_UNNAMED if the variable is a system-generate name.
OUTVARn	Description of the n-th output variable for the SELECT statement. The comma-separated fields follow the same convention as in INVARs.

Related concepts:

- “Introduction to DB2 CLI and ODBC” on page 3

Related tasks:

- “Creating static SQL with CLI/ODBC/JDBC Static Profiling” on page 185

Chapter 22. CLI/ODBC/JDBC trace facility

CLI/ODBC/JDBC trace facility

This topic discusses the following subjects:

- “DB2 CLI and DB2 JDBC trace configuration”
- “DB2 CLI trace options and the db2cli.ini file” on page 190
- “DB2 JDBC trace options and the db2cli.ini file” on page 191
- “DB2 CLI driver trace versus ODBC driver manager trace” on page 192
- “DB2 CLI driver, DB2 JDBC Type 2 Driver, and DB2 traces” on page 193
- “DB2 CLI and DB2 JDBC traces and CLI or Java stored procedures” on page 193

The DB2 CLI and the DB2 JDBC Type 2 Driver for Linux, UNIX, and Windows offer comprehensive tracing facilities. By default, these facilities are disabled and use no additional computing resources. When enabled, the trace facilities generate one or more text log files whenever an application accesses the appropriate driver (DB2 CLI or DB2 JDBC Type 2 Driver). These log files provide detailed information about:

- the order in which CLI or JDBC functions were called by the application
- the contents of input and output parameters passed to and received from CLI or JDBC functions
- the return codes and any error or warning messages generated by CLI or JDBC functions

Note: This trace facility does not apply to the DB2 Universal JDBC Driver.

DB2 CLI and DB2 JDBC trace file analysis can benefit application developers in a number of ways. First, subtle program logic and parameter initialization errors are often evident in the traces. Second, DB2 CLI and DB2 JDBC traces might suggest ways of better tuning an application or the databases it accesses. For example, if a DB2 CLI trace shows a table being queried many times on a particular set of attributes, an index corresponding to those attributes might be created on the table to improve application performance. Finally, analysis of DB2 CLI and DB2 JDBC trace files can help application developers understand how a third party application or interface is behaving.

DB2 CLI and DB2 JDBC trace configuration:

The configuration parameters for both DB2 CLI and DB2 JDBC traces facilities are read from the DB2 CLI configuration file `db2cli.ini`. By default, this file is located in the `\sql11ib` path on the Windows platform and the `/sql11ib/cfg` path on UNIX platforms. You can override the default path by setting the `DB2CLIINIPATH` environment variable. On the Windows platform, an additional `db2cli.ini` file can be found in the user's profile (or home) directory if there are any user-defined data sources defined using the ODBC Driver Manager. This `db2cli.ini` file will override the default file.

To view the current `db2cli.ini` trace configuration parameters from the command line processor, issue the following command:

```
db2 GET CLI CFG FOR SECTION COMMON
```

There are three ways to modify the `db2cli.ini` file to configure the DB2 CLI and DB2 JDBC trace facilities:

- use the DB2 Configuration Assistant if it is available
- manually edit the `db2cli.ini` file using a text editor
- issue the `UPDATE CLI CFG` command from the command line processor

For example, the following command issued from the command line processor updates the `db2cli.ini` file and enables the JDBC tracing facility:

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING jdbctrace 1
```

Notes:

1. Typically the DB2 CLI and DB2 JDBC trace configuration options are only read from the `db2cli.ini` configuration file at the time an application is initialized. However, a special `db2cli.ini` trace option, `TraceRefreshInterval`, can be used to indicate an interval at which specific DB2 CLI trace options are reread from the `db2cli.ini` file.
2. The DB2 CLI tracing facility can also be configured programmatically by setting the `SQL_ATTR_TRACE` environment attribute. This setting will override the settings contained in the `db2cli.ini` file.

Important: Disable the DB2 CLI and DB2 JDBC trace facilities when they are not needed. Unnecessary tracing can reduce application performance and generate unwanted trace log files. DB2 does not delete any generated trace files and will append new trace information to any existing trace files.

DB2 CLI Trace options and the `db2cli.ini` file:

When an application using the DB2 CLI driver begins execution, the driver checks for trace facility options in the `[COMMON]` section of the `db2cli.ini` file. These trace options are specific trace keywords that are set to certain values in the `db2cli.ini` file under the `[COMMON]` section.

Note: Because DB2 CLI trace keywords appear in the `[COMMON]` section of the `db2cli.ini` file, their values apply to all database connections through the DB2 CLI driver.

The DB2 CLI trace keywords that can be defined are:

- `Trace`
- `TraceComm`
- `TraceErrImmediateTraceErrImmediate`
- `TraceFileName`
- `TraceFlush`
- `TraceFlushOnErrorTraceFlushOnError`
- `TraceLocks`
- `TracePathName`
- `TracePIDList`
- `TracePIDTID`
- `TraceRefreshInterval`
- `TraceStmtOnly`
- `TraceTime`
- `TraceTimeStamp`

Note: DB2 CLI trace keywords are only read from the `db2cli.ini` file once at application initialization time unless the `TraceRefreshInterval` keyword is set. If this keyword is set, the `Trace` and `TracePIDList` keywords are reread from the `db2cli.ini` file at the specified interval and applied, as appropriate, to the currently executing application.

An example `db2cli.ini` file trace configuration using these DB2 CLI keywords and values is:

```
[COMMON]
trace=1
TraceFileName=\temp\clitrace.txt
TraceFlush=1
```

Notes:

1. CLI trace keywords are NOT case sensitive. However, path and file name keyword values might be case-sensitive on some operating systems (such as UNIX).
2. If either a DB2 CLI trace keyword or its associated value in the `db2cli.ini` file is invalid, the DB2 CLI trace facility will ignore it and use the default value for that trace keyword instead.

DB2 JDBC Trace options and the `db2cli.ini` file:

When an application using the DB2 JDBC Type 2 Driver begins execution, the driver also checks for trace facility options in the `db2cli.ini` file. As with the DB2 CLI trace options, DB2 JDBC trace options are specified as keyword/value pairs located under the `[COMMON]` section of the `db2cli.ini` file.

Note: Because DB2 JDBC trace keywords appear in the `[COMMON]` section of the `db2cli.ini` file, their values apply to all database connections through the DB2 JDBC Type 2 Driver.

The DB2 JDBC trace keywords that can be defined are:

- `JDBCTrace`
- `JDBCTracePathName`
- `JDBCTraceFlush`

`JDBCTrace = 0 | 1`

The `JDBCTrace` keyword controls whether or not other DB2 JDBC tracing keywords have any effect on program execution. Setting `JDBCTrace` to its default value of 0 disables the DB2 JDBC trace facility. Setting `JDBCTrace` to 1 enables it.

By itself, the `JDBCTrace` keyword has little effect and produces no trace output unless the `JDBCTracePathName` keyword is also specified.

`JDBCTracePathName = <fully_qualified_trace_path_name>`

The value of `JDBCTracePathName` is the fully qualified path of the directory to which all DB2 JDBC trace information is written. The DB2 JDBC trace facility attempts to generate a new trace log file each time a JDBC application is executed using the DB2 JDBC Type 2 Driver. If the application is multithreaded, a separate trace log file will be generated for each thread. A concatenation of the application process ID, the thread sequence number, and a thread-identifying string are automatically used to name trace log files. There is no default path name to which DB2 JDBC trace output log files are written.

JDBCTraceFlush = 0 | 1

The JDBCTraceFlush keyword specifies how often trace information is written to the DB2 JDBC trace log file. By default, JDBCTraceFlush is set to 0 and each DB2 JDBC trace log file is kept open until the traced application or thread terminates normally. If the application terminates abnormally, some trace information that was not written to the trace log file might be lost.

To ensure the integrity and completeness of the trace information written to the DB2 JDBC trace log file, the JDBCTraceFlush keyword can be set to 1. After each trace entry has been written to the trace log file, the DB2 JDBC driver closes the file and then reopens it, appending new trace entries to the end of the file. This guarantees that no trace information will be lost.

Note: *Each DB2 JDBC log file close and reopen operation incurs significant input/output overhead and can reduce application performance considerably.*

An example db2cli.ini file trace configuration using these DB2 JDBC keywords and values is:

```
[COMMON]
jdbctrace=1
JdbcTracePathName=\temp\jdbctrace\
JDBCTraceFlush=1
```

Notes:

1. JDBC trace keywords are NOT case sensitive. However, path and file name keyword values might be case-sensitive on some operating systems (such as UNIX).
2. If either a DB2 JDBC trace keyword or its associated value in the db2cli.ini file is invalid, the DB2 JDBC trace facility will ignore it and use the default value for that trace keyword instead.
3. Enabling DB2 JDBC tracing does not enable DB2 CLI tracing. The DB2 JDBC Type 2 Driver depends on the DB2 CLI driver to access the database. Consequently, Java™ developers might also want to enable DB2 CLI tracing for additional information on how their applications interact with the database through the various software layers. DB2 JDBC and DB2 CLI trace options are independent of each other and can be specified together in any order under the [COMMON] section of the db2cli.ini file.

DB2 CLI Driver trace versus ODBC driver manager trace:

It is important to understand the difference between an ODBC driver manager trace and a DB2 CLI driver trace. An ODBC driver manager trace shows the ODBC function calls made by an ODBC application to the ODBC driver manager. In contrast, a DB2 CLI driver trace shows the function calls made by the ODBC driver manager to the DB2 CLI driver *on behalf of the application*.

An ODBC driver manager might forward some function calls directly from the application to the DB2 CLI driver. However, the ODBC driver manager might also delay or avoid forwarding some function calls to the driver. The ODBC driver manager might also modify application function arguments or map application functions to other functions before forwarding the call on to the DB2 CLI driver.

Reasons for application function call intervention by the ODBC driver manager include:

- Applications written using ODBC 2.0 functions that have been deprecated in ODBC 3.0 will have the old functions mapped to new functions.
- ODBC 2.0 function arguments deprecated in ODBC 3.0 will be mapped to equivalent ODBC 3.0 arguments.
- The Microsoft cursor library will map calls such as `SQLExtendedFetch()` to multiple calls to `SQLFetch()` and other supporting functions to achieve the same end result.
- ODBC driver manager connection pooling will usually defer `SQLDisconnect()` requests (or avoid them altogether if the connection gets reused).

For these and other reasons, application developers might find an ODBC driver manager trace to be a useful complement to the DB2 CLI driver trace.

For more information on capturing and interpreting ODBC driver manager traces, refer to the ODBC driver manager documentation. On the Windows platforms, refer to the Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, also available online at: <http://www.msdn.microsoft.com/>.

DB2 CLI Driver, DB2 JDBC Type 2 Driver, and DB2 traces:

Internally, the DB2 JDBC Type 2 Driver makes use of the DB2 CLI driver for database access. For example, the Java `getConnection()` method is internally mapped by the DB2 JDBC Type 2 Driver to the DB2 CLI `SQLConnect()` function. As a result, Java developers might find a DB2 CLI trace to be a useful complement to the DB2 JDBC trace.

The DB2 CLI driver makes use of many internal and DB2 specific functions to do its work. These internal and DB2 specific function calls are logged in the DB2 trace. Application developers will not find DB2 traces useful, as they are only meant to assist IBM Service in problem determination and resolution.

DB2 CLI and DB2 JDBC traces and CLI or Java stored procedures:

On all workstation platforms, the DB2 CLI and DB2 JDBC trace facilities can be used to trace DB2 CLI and DB2 JDBC stored procedures.

Most of the DB2 CLI and DB2 JDBC trace information and instructions given in earlier sections is generic and applies to both applications and stored procedures equally. However, unlike applications which are clients of a database server (and typically execute on a machine separate from the database server), stored procedures execute at the database server. Therefore, the following additional steps must be taken when tracing DB2 CLI or DB2 JDBC stored procedures:

- Ensure the trace keyword options are specified in the `db2cli.ini` file located at the DB2 server.
- If the `TraceRefreshInterval` keyword is not set to a positive, non-zero value, ensure all keywords are configured correctly prior to database startup time (that is, when the `db2start` command is issued). Changing trace settings while the database server is running can cause unpredictable results. For example, if the `TracePathName` is changed while the server is running, then the next time a stored procedure is executed, some trace files might be written to the new path, while others are written to the original path. To ensure consistency, restart the server any time a trace keyword other than `Trace` or `TracePIDList` is modified.

Related concepts:

- "CLI and JDBC trace files" on page 194

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “db2trc - Trace command” in *Command Reference*
- “GET CLI CONFIGURATION command” in *Command Reference*
- “Miscellaneous variables” in *Performance Guide*
- “SQLSetEnvAttr function (CLI) - Set environment attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “UPDATE CLI CONFIGURATION command” in *Command Reference*

CLI and JDBC trace files

Applications that access the DB2 CLI and DB2 JDBC drivers can make use of the DB2 CLI and DB2 JDBC trace facilities. These utilities record all function calls made by the DB2 CLI or DB2 JDBC drivers to a log file which is useful for problem determination. This topic discusses how to access and interpret these log files generated by the tracing facilities:

- “CLI and JDBC trace file location”
- “CLI trace file interpretation” on page 195
- “JDBC trace file interpretation” on page 199

CLI and JDBC trace file location:

If the TraceFileName keyword was used in the db2cli.ini file to specify a fully qualified file name, then the DB2 CLI trace log file will be in the location specified. If a relative file name was specified for the DB2 CLI trace log file name, the location of that file will depend on what the operating system considers to be the current path of the application.

Note: If the user executing the application does not have sufficient authority to write to the trace log file in the specified path, no file will be generated and no warning or error is given.

If either or both of the TracePathName and JDBCTracePathName keywords were used in the db2cli.ini file to specify fully qualified directories, then the DB2 CLI and DB2 JDBC trace log files will be in the location specified. If a relative directory name was specified for either or both trace directories, the operating system will determine its location based on what it considers to be the current path of the application.

Note: If the user executing the application does not have sufficient authority to write trace files in the specified path, no file will be generated and no warning or error is given. If the specified trace path does not exist, it will not be created.

The DB2 CLI and DB2 JDBC trace facilities automatically use the application’s process ID and thread sequence number to name the trace log files when the TracePathName and JDBCTracePathName keywords have been set. For example, a DB2 CLI trace of an application with three threads might generate the following DB2 CLI trace log files: 100390.0, 100390.1, 100390.2.

Similarly, a DB2 JDBC trace of a Java application with two threads might generate the following JDBC trace log files: 7960main.trc, 7960Thread-1.trc.

Note: If the trace directory contains both old and new trace log files, file date and time stamp information can be used to locate the most recent trace files.

If no DB2 CLI or DB2 JDBC trace output files appear to have been created:

- Verify that the trace configuration keywords are set correctly in the `db2cli.ini` file. Issuing the `db2 GET CLI CFG FOR SECTION COMMON` command from the command line processor is a quick way to do this.
- Ensure the application is restarted after updating the `db2cli.ini` file. Specifically, the DB2 CLI and DB2 JDBC trace facilities are initialized during application startup. Once initialized, the DB2 JDBC trace facility cannot be reconfigured. The DB2 CLI trace facility can be reconfigured at run time but only if the `TraceRefreshInterval` keyword was appropriately specified prior to application startup.

Note: Only the `Trace` and `TracePIDList` DB2 CLI keywords can be reconfigured at run time. *Changes made to other DB2 CLI keywords, including `TraceRefreshInterval`, have no effect without an application restart.*

- If the `TraceRefreshInterval` keyword was specified prior to application startup, and if the `Trace` keyword was initially set to 0, ensure that enough time has elapsed for the DB2 CLI trace facility to reread the `Trace` keyword value.
- If either or both the `TracePathName` and `JDBCTracePathName` keywords are used to specify trace directories, ensure those directories exist prior to starting the application.
- Ensure the application has write access to the specified trace log file or trace directory.
- Check the `DB2CLIINIPATH` environment variable. If set, the DB2 CLI and DB2 JDBC trace facilities expect the `db2cli.ini` file to be at the location specified by this variable.
- If the application uses ODBC to interface with the DB2 CLI driver, verify that one of the `SQLConnect()`, `SQLDriverConnect()` or `SQLBrowseConnect()` functions have been successfully called. No entries will be written to the DB2 CLI trace log files until a database connection has successfully been made.

CLI trace file interpretation:

DB2 CLI traces always begin with a header that identifies the process ID and thread ID of the application that generated the trace, the time the trace began, and product specific information such as the local DB2 build level and DB2 CLI driver version. For example:

```
1 [ Process: 1227, Thread: 1024 ]
2 [ Date, Time:          01-27-2002 13:46:07.535211 ]
3 [ Product:            QDB2/LINUX 7.1.0 ]
4 [ Level Identifier:   02010105 ]
5 [ CLI Driver Version: 07.01.0000 ]
6 [ Informational Tokens: "DB2 v7.1.0","n000510", "" ]
```

Note: Trace examples used in this section have line numbers added to the left hand side of the trace. These line numbers have been added to aid the discussion and will *not* appear in an actual DB2 CLI trace.

Immediately following the trace header, there are usually a number of trace entries related to environment and connection handle allocation and initialization. For example:

```

7  SQLAllocEnv( phEnv=&bffff684 )
8  → Time elapsed - +9.200000E-004 seconds

9  SQLAllocEnv( phEnv=0:1 )
10 ← SQL_SUCCESS Time elapsed - +7.500000E-004 seconds

11 SQLAllocConnect( hEnv=0:1, phDbc=&bffff680 )
12 → Time elapsed - +2.334000E-003 seconds

13 SQLAllocConnect( phDbc=0:1 )
14 ← SQL_SUCCESS Time elapsed - +5.280000E-004 seconds

15 SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
16 → Time elapsed - +2.301000E-003 seconds

17 SQLSetConnectOption( )
18 ← SQL_SUCCESS Time elapsed - +3.150000E-004 seconds

19 SQLConnect( hDbc=0:1, szDSN="SAMPLE", cbDSN=-3, szUID="", cbUID=-3,
              szAuthStr="", cbAuthStr=-3 )
20 → Time elapsed - +7.000000E-005 seconds
21 ( DBMS NAME="DB2/LINUX", Version="07.01.0000", Fixpack="0x22010105" )

22 SQLConnect( )
23 ← SQL_SUCCESS Time elapsed - +5.209880E-001 seconds
24 ( DSN="SAMPLE" )

25 ( UID=" " )

26 ( PWD="*" )

```

In the above trace example, notice that there are two entries for each DB2 CLI function call (for example, lines 19-21 and 22-26 for the SQLConnect() function call). This is always the case in DB2 CLI traces. The first entry shows the input parameter values passed to the function call while the second entry shows the function output parameter values and return code returned to the application.

The above trace example shows that the SQLAllocEnv() function successfully allocated an environment handle (phEnv=0:1) at line 9. That handle was then passed to the SQLAllocConnect() function which successfully allocated a database connection handle (phDbc=0:1) as of line 13. Next, the SQLSetConnectOption() function was used to set the phDbc=0:1 connection's SQL_ATTR_AUTOCOMMIT attribute to SQL_AUTOCOMMIT_OFF (vParam=0) at line 15. Finally, SQLConnect() was called to connect to the target database (SAMPLE) at line 19.

Included in the input trace entry of the SQLConnect() function on line 21 is the build and FixPak level of the target database server. Other information that might also appear in this trace entry includes input connection string keywords and the code pages of the client and server. For example, suppose the following information also appeared in the SQLConnect() trace entry:

```

( Application Codepage=819, Database Codepage=819,
  Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=819,
  Application Char Codepage=819, Application Graphic Codepage=819 )

```

This would mean the application and the database server were using the same code page (819).

The return trace entry of the `SQLConnect()` function also contains important connection information (lines 24-26 in the above example trace). Additional information that might be displayed in the return entry includes any `PATCH1` or `PATCH2` keyword values that apply to the connection. For example, if `PATCH2=27,28` was specified in the `db2cli.ini` file under the `COMMON` section, the following line should also appear in the `SQLConnect()` return entry:

```
( PATCH2="27,28" )
```

Following the environment and connection related trace entries are the statement related trace entries. For example:

```
27  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
28      —> Time elapsed - +1.868000E-003 seconds

29  SQLAllocStmt( phStmt=1:1 )
30      <— SQL_SUCCESS   Time elapsed - +6.890000E-004 seconds

31  SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                                     VARCHAR(10))", cbSqlStr=-3 )
32      —> Time elapsed - +2.863000E-003 seconds
33  ( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

34  SQLExecDirect( )
35      <— SQL_SUCCESS   Time elapsed - +2.387800E-002 seconds
```

In the above trace example, the database connection handle (`phDbc=0:1`) was used to allocate a statement handle (`phStmt=1:1`) at line 29. An unprepared SQL statement was then executed on that statement handle at line 31. If the `TraceComm=1` keyword had been set in the `db2cli.ini` file, the `SQLExecDirect()` function call trace entries would have shown additional client-server communication information as follows:

```
SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                                     VARCHAR(10))", cbSqlStr=-3 )
      —> Time elapsed - +2.876000E-003 seconds
( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

    sqlccsend( ulBytes - 232 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.150000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.243800E-002

SQLExecDirect( )
    <— SQL_SUCCESS   Time elapsed - +2.384900E-002 seconds
```

Notice the additional `sqlccsend()` and `sqlccrecv()` function call information in this trace entry. The `sqlccsend()` call information reveals how much data was sent from the client to the server, how long the transmission took, and the success of that transmission (`0 = SQL_SUCCESS`). The `sqlccrecv()` call information then reveals how long the client waited for a response from the server and the amount of data included in the response.

Often, multiple statement handles will appear in the DB2 CLI trace. By paying close attention to the statement handle identifier, one can easily follow the execution path of a statement handle independent of all other statement handles appearing in the trace.

Statement execution paths appearing in the DB2 CLI trace are usually more complicated than the example shown above. For example:

```

36 SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
37     → Time elapsed - +1.532000E-003 seconds

38 SQLAllocStmt( phStmt=1:2 )
39     ← SQL_SUCCESS   Time elapsed - +6.820000E-004 seconds

40 SQLPrepare( hStmt=1:2, pszSqlStr="INSERT INTO GREETING VALUES ( ? )",
              cbSqlStr=-3 )
41     → Time elapsed - +2.733000E-003 seconds
42 ( StmtOut="INSERT INTO GREETING VALUES ( ? )" )

43 SQLPrepare( )
44     ← SQL_SUCCESS   Time elapsed - +9.150000E-004 seconds

45 SQLBindParameter( hStmt=1:2, iPar=1, fParamType=SQL_PARAM_INPUT,
                   fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=14,
                   ibScale=0, rgbValue=&080eca70, cbValueMax=15,
                   pcbValue=&080eca4c )
46     → Time elapsed - +4.091000E-003 seconds

47 SQLBindParameter( )
48     ← SQL_SUCCESS   Time elapsed - +6.780000E-004 seconds

49 SQLExecute( hStmt=1:2 )
50     → Time elapsed - +1.337000E-003 seconds
51 ( iPar=1, fCType=SQL_C_CHAR, rgbValue="Hello World!!!", pcbValue=14,
    piIndicatorPtr=14 )

52 SQLExecute( )
53     ← SQL_ERROR     Time elapsed - +5.951000E-003 seconds

```

In the above trace example, the database connection handle (`phDbc=0:1`) was used to allocate a second statement handle (`phStmt=1:2`) at line 38. An SQL statement with one parameter marker was then prepared on that statement handle at line 40. Next, an input parameter (`iPar=1`) of the appropriate SQL type (`SQL_CHAR`) was bound to the parameter marker at line 45. Finally, the statement was executed at line 49. Notice that both the contents and length of the input parameter (`rgbValue="Hello World!!!"`, `pcbValue=14`) are displayed in the trace on line 51.

The `SQLExecute()` function fails at line 52. If the application calls a diagnostic DB2 CLI function like `SQLError()` to diagnose the cause of the failure, then that cause will appear in the trace. For example:

```

54 SQLError( hEnv=0:1, hDbc=0:1, hStmt=1:2, pszSqlState=&bffff680,
            pfNativeError=&bffffee78, pszErrorMsg=&bffff280,
            cbErrorMsgMax=1024, pcbErrorMsg=&bffffee76 )
55     → Time elapsed - +1.512000E-003 seconds

56 SQLError( pszSqlState="22001", pfNativeError=-302, pszErrorMsg="[IBM][CLI
  Driver][DB2/LINUX] SQL0302N The value of a host variable in the EXECUTE
  or OPEN statement is too large for its corresponding use.
  SQLSTATE=22001", pcbErrorMsg=157 )
57     ← SQL_SUCCESS   Time elapsed - +8.060000E-004 seconds

```

The error message returned at line 56 contains the DB2 native error code that was generated (`SQL0302N`), the `sqlstate` that corresponds to that code (`SQLSTATE=22001`) and a brief description of the error. In this example, the source of the error is evident: on line 49, the application is trying to insert a string with 14 characters into a column defined as `VARCHAR(10)` on line 31.

If the application does not respond to a DB2 CLI function warning or error return code by calling a diagnostic function like `SQLError()`, the warning or error message

should still be written to the DB2 CLI trace. However, the location of that message in the trace may not be close to where the error actually occurred. Furthermore, the trace will indicate that the error or warning message was not retrieved by the application. For example, if not retrieved, the error message in the above example might not appear until a later, seemingly unrelated DB2 CLI function call as follows:

```

SQLDisconnect( hDbc=0:1 )
    —> Time elapsed - +1.501000E-003 seconds
        sqlccsend( ulBytes - 72 )
        sqlccsend( Handle - 1084869448 )
        sqlccsend( ) - rc - 0, time elapsed - +1.080000E-004
        sqlccrecv( )
        sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +1.717950E-001
( Unretrieved error message="SQL0302N The value of a host variable in the
EXECUTE or OPEN statement is too large for its corresponding use.
SQLSTATE=22001" )

SQLDisconnect( )
    <— SQL_SUCCESS Time elapsed - +1.734130E-001 seconds

```

The final part of a DB2 CLI trace should show the application releasing the database connection and environment handles that it allocated earlier in the trace. For example:

```

58 SQLTransact( hEnv=0:1, hDbc=0:1, fType=SQL_ROLLBACK )
59     —> Time elapsed - +6.085000E-003 seconds
60 ( ROLLBACK=0 )

61 SQLTransact( )
    <— SQL_SUCCESS Time elapsed - +2.220750E-001 seconds

62 SQLDisconnect( hDbc=0:1 )
63     —> Time elapsed - +1.511000E-003 seconds

64 SQLDisconnect( )
65     <— SQL_SUCCESS Time elapsed - +1.531340E-001 seconds

66 SQLFreeConnect( hDbc=0:1 )
67     —> Time elapsed - +2.389000E-003 seconds

68 SQLFreeConnect( )
69     <— SQL_SUCCESS Time elapsed - +3.140000E-004 seconds

70 SQLFreeEnv( hEnv=0:1 )
71     —> Time elapsed - +1.129000E-003 seconds

72 SQLFreeEnv( )
73     <— SQL_SUCCESS Time elapsed - +2.870000E-004 seconds

```

JDDBC trace file interpretation:

DB2 JDDBC traces always begin with a header that lists important system information such as key environment variable settings, the SDK for Java or JRE level, the DB2 JDDBC driver level, and the DB2 build level. For example:

```

1  =====
2  | Trace beginning on 2002-1-28 7:21:0.19
3  =====

4  System Properties:
5  -----
6  user.language = en
7  java.home = c:\Program Files\SQLLIB\java\jdk\bin\..
8  java.vendor.url.bug =
9  awt.toolkit = sun.awt.windows.WToolkit

```

```

10 file.encoding.pkg = sun.io
11 java.version = 1.1.8
12 file.separator = \
13 line.separator =
14 user.region = US
15 file.encoding = Cp1252
16 java.compiler = ibmjtc
17 java.vendor = IBM Corporation
18 user.timezone = EST
19 user.name = db2user
20 os.arch = x86
21 java.fullversion = JDK 1.1.8 IBM build n118p-19991124 (JIT ibmjtc
      V3.5-IBMJDK1.1-19991124)
22 os.name = Windows NT
23 java.vendor.url = http://www.ibm.com/
24 user.dir = c:\Program Files\SQLLIB\samples\java
25 java.class.path =
      .:C:\Program Files\SQLLIB\lib;C:\Program Files\SQLLIB\java;
      C:\Program Files\SQLLIB\java\jdk\bin\
26 java.class.version = 45.3
27 os.version = 5.0
28 path.separator = ;
29 user.home = C:\home\db2user
30 -----

```

Note: Trace examples used in this section have line numbers added to the left hand side of the trace. These line numbers have been added to aid the discussion and will *not* appear in an actual DB2 JDBC trace.

Immediately following the trace header, one usually finds a number of trace entries related to initialization of the JDBC environment and database connection establishment. For example:

```

31 jdbc.app.DB2Driver -> DB2Driver() (2002-1-28 7:21:0.29)
32 | Loaded db2jdbc from java.library.path
33 jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.01]

34 DB2Driver - connect(jdbc:db2:sample)

35 jdbc.app.DB2ConnectionTrace -> connect( sample, info, db2driver, 0, false )
      (2002-1-28 7:21:0.59)
36 | 10: connectionHandle = 1
37 jdbc.app.DB2ConnectionTrace <- connect() [Time Elapsed = 0.16]

38 jdbc.app.DB2ConnectionTrace -> DB2Connection (2002-1-28 7:21:0.219)
39 | source = sample
40 | Connection handle = 1
41 jdbc.app.DB2ConnectionTrace <- DB2Connection

```

In the above trace example, a request to load the DB2 JDBC driver was made on line 31. This request returned successfully as reported on line 33.

The DB2 JDBC trace facility uses specific Java classes to capture the trace information. In the above trace example, one of those trace classes, DB2ConnectionTrace, has generated two trace entries numbered 35-37 and 38-41.

Line 35 shows the connect() method being invoked and the input parameters to that method call. Line 37 shows that the connect() method call has returned successfully while line 36 shows the output parameter of that call (Connection handle = 1).

Following the connection related entries, one usually finds statement related entries in the JDBC trace. For example:

```

42 jdbc.app.DB2ConnectionTrace -> createStatement() (2002-1-28 7:21:0.219)
43 | Connection handle = 1
44 | jdbc.app.DB2StatementTrace -> DB2Statement( con, 1003, 1007 )
    | (2002-1-28 7:21:0.229)
45 | jdbc.app.DB2StatementTrace <- DB2Statement() [Time Elapsed = 0.0]
46 | jdbc.app.DB2StatementTrace -> DB2Statement (2002-1-28 7:21:0.229)
47 | | Statement handle = 1:1
48 | jdbc.app.DB2StatementTrace <- DB2Statement
49 jdbc.app.DB2ConnectionTrace <- createStatement - Time Elapsed = 0.01

50 jdbc.app.DB2StatementTrace -> executeQuery(SELECT * FROM EMPLOYEE WHERE
    empno = 000010) (2002-1-28 7:21:0.269)
51 | Statement handle = 1:1
52 | jdbc.app.DB2StatementTrace -> execute2( SELECT * FROM EMPLOYEE WHERE
    empno = 000010 ) (2002-1-28 7:21:0.269)
52 | | | jdbc.DB2Exception -> DB2Exception() (2002-1-28 7:21:0.729)
53 | | | | 10: SQLError = [IBM][CLI Driver][DB2/NT] SQL0401N The data types of
    | | | | the operands for the operation "=" are not compatible.
    | | | | SQLSTATE=42818
54 | | | | | SQLState = 42818
55 | | | | | SQLNativeCode = -401
56 | | | | | LineNumber = 0
57 | | | | | SQLerrmc = =
58 | | | | | jdbc.DB2Exception <- DB2Exception() [Time Elapsed = 0.0]
59 | | | | | jdbc.app.DB2StatementTrace <- executeQuery - Time Elapsed = 0.0

```

On line 42 and 43, the DB2ConnectionTrace class reported that the JDBC createStatement() method had been called with connection handle 1. Within that method, the internal method DB2Statement() was called as reported by another DB2 JDBC trace facility class, DB2StatementTrace. Notice that this internal method call appears 'nested' in the trace entry. Lines 47-49 show that the methods returned successfully and that statement handle 1:1 was allocated.

On line 50, an SQL query method call is made on statement 1:1, but the call generates an exception at line 52. The error message is reported on line 53 and contains the DB2 native error code that was generated (SQL0401N), the sqlstate that corresponds to that code (SQLSTATE=42818) and a brief description of the error. In this example, the error results because the EMPLOYEE.EMPNO column is defined as CHAR(6) and not an integer value as assumed in the query.

Related concepts:

- "CLI/ODBC/JDBC trace facility" on page 189

Related reference:

- "Miscellaneous variables" in *Performance Guide*
- "Trace CLI/ODBC configuration keyword" on page 375
- "TraceComm CLI/ODBC configuration keyword" on page 376
- "TraceFileName CLI/ODBC configuration keyword" on page 378
- "TracePathName CLI/ODBC configuration keyword" on page 381
- "TracePIDList CLI/ODBC configuration keyword" on page 382
- "TraceRefreshInterval CLI/ODBC configuration keyword" on page 383

Chapter 23. CLI bind files and package names

DB2 CLI bind files and package names

DB2 CLI packages are automatically bound to databases when the databases are created or migrated. If a fix pack is applied to either the client or the server, or a user has intentionally dropped a package, then you must rebind `db2cli.lst` by issuing the following command:

UNIX

```
db2 bind <BNDPATH>/db2cli.lst blocking all grant public
```

Windows

```
db2 bind "%DB2PATH%\bnd\db2cli.lst" blocking all grant public
```

The `db2cli.lst` file contains the names of the required bind files for DB2 CLI to connect to DB2 servers on Linux, UNIX, and Windows (`db2clipk.bnd` and `db2clist.bnd`).

For host and iSeries servers use one of `ddcsvm.lst`, `ddcsmvs.lst`, `ddcsvse.lst`, or `ddcs400.lst` bind list files.

Warnings that are generated when binding DB2 CLI packages (such as `db2clist.bnd` or `db2cli.lst`) to workstation or host servers are expected. This is because DB2 uses generic bind files, but the bind file packages for DB2 CLI packages contain sections that apply to specific platforms. Therefore, DB2 might generate warnings during the binding against a server, when it encounters a platform-specific section that does not apply to the server.

The following is an example of a warning that can be ignored which might occur when binding a DB2 CLI package (such as `db2clist.bnd` or `db2cli.lst`) to a workstation server:

```
LINE      MESSAGES FOR db2clist.bnd
-----
235      SQL0440N  No authorized routine named "POSSTR" of type
          "FUNCTION" having compatible arguments was found.
          SQLSTATE=42884
```

Table 18. DB2 CLI Bind files and package names

Bind file name	Package name	Needed by DB2 servers on Linux, UNIX, and Windows	Needed by host servers	Description
db2clipk.bnd	SYSSHxyy	Yes	Yes	dynamic placeholders - small package WITH HOLD
	SYSSNxyy	Yes	Yes	dynamic placeholders - small Package NOT WITH HOLD
	SYSLHxyy	Yes	Yes	dynamic placeholders - large package WITH HOLD
	SYSLNxyy	Yes	Yes	dynamic placeholders - large package NOT WITH HOLD
db2clist.bnd	SYSSTAT	Yes	Yes	common static CLI functions
db2schema.bnd	SQLL9vyv	Yes	No	catalog function support

Note:

- 'S' represents a small package and 'L' represents a large package
- 'H' represents WITH HOLD, and 'N' represents NOT WITH HOLD.
- 'v' represents the DB2 server version: for example, E=Version 8, F=Version 9
- 'x' is the isolation level: 0=NC, 1=UR, 2=CS, 3=RS, 4=RR
- 'yy' is the package iteration 00 through FF
- 'zz' is unique for each platform

For example, for the dynamic packages:

- SYSSN100 A small package (65 sections) where all cursor declarations are for non-held cursors. Bound with isolation level UR. This is the first iteration of that package.
- SYSLH401 A large package (385 sections) where all cursor declarations are for held cursors. Bound with isolation level RS. This is the second iteration of that package.

Previous versions of DB2 servers do not need all of the bind files and will therefore return errors at bind time. Use the bind option `SQLERROR(CONTINUE)` so that the same package can be bound on all platforms and errors will be ignored for any statements not supported there.

db2schema.bnd bind file:

The `db2schema.bnd` bind file is automatically bound when the database is created or migrated on DB2 servers on Linux, UNIX, and Windows, and exists only on these types of servers. This bind file is located at the server and should be bound manually (from the server) if a fix pack was applied on the server, if the package

was intentionally dropped by a user or if an SQL1088W (+1088) warning is received after database creation or migration.

Only the most recent version of this package is needed.

If the package is missing, it must be rebound locally on the server. Do not bind this package against remote servers (for example, against a host database). The bind file is found in the sql11b/bnd directory of the instance home directory, and is rebound with the following command:

```
bind db2schema.bnd blocking all grant public
```

If an SQL1088W warning was received after database creation or migration, and the db2schema.bnd package is missing, increase the APPLHEAPSZ database configuration parameter to 128 or greater, and attempt to rebound. No errors should be reported during binding.

Related concepts:

- “Packages” in *SQL Reference, Volume 1*

Related tasks:

- “Setting up the CLI environment” on page 209

Related reference:

- “BIND command” in *Command Reference*

Bind option limitations for CLI packages

Some bind options might not take effect when binding DB2 CLI packages with any of the following list files: db2cli.lst, ddcsmvs.lst, ddcs400.lst, ddcsvm.lst, or ddcsvse.lst. Because DB2 CLI packages are used by DB2 CLI, ODBC, JDBC, OLE DB, .NET, and ADO applications, any changes made to the DB2 CLI packages affect all applications of these types. Only a subset of bind options are therefore supported by default when binding DB2 CLI packages. The supported options are: ACTION, COLLECTION, CLIPKG, OWNER, and REPLVER. All other bind options that impact CLI packages are ignored.

To create DB2 CLI packages with bind options that are not supported by default, specify the COLLECTION bind option with a collection ID that is different from the default collection ID, NULLID. Any bind options specified are then accepted. For example, to create DB2 CLI packages with the KEEP DYNAMIC YES bind option, which is not supported by default, issue the following command:

```
db2 bind @db2cli.lst collection newcolid keepdynamic yes
```

In order for CLI/ODBC applications to access the DB2 CLI packages created in the new collection, set the CurrentPackageSet CLI/ODBC keyword in the db2cli.ini initialization file to the new collection ID.

To overwrite DB2 CLI packages that already exist under a particular collection ID, perform either of the following actions:

- Drop the existing CLI package before issuing the bind command for this collection ID.
- Specify the ACTION REPLACE bind option when issuing the bind command.

Related reference:

- “CurrentPackageSet CLI/ODBC configuration keyword” on page 305
- “DB2 CLI bind files and package names” on page 203
- “BIND command” in *Command Reference*

Part 3. CLI environment and application building

Chapter 24. CLI environmental setup

Setting up the CLI environment	209	Setting up the Windows CLI environment	215
Setting up the UNIX ODBC environment	210	Selecting a different DB2 copy for your Windows	
Setting up the unixODBC Driver Manager.	212	CLI application.	217
Sample build scripts and configurations for the			
unixODBC Driver Manager.	213		

Before you can run a CLI application, you must set up the CLI environment. This chapter describes how to set up the CLI or ODBC environment on the UNIX and Windows platforms.

Setting up the CLI environment

Runtime support for DB2 CLI applications is contained in all DB2 clients. Support for building and running DB2 CLI applications is contained in the DB2 Client. This section describes the general setup required for DB2 CLI runtime support.

Prerequisites:

Before you set up your CLI environment, ensure you have set up the application development environment.

Procedure:

In order for a DB2 CLI application to successfully access a DB2 database:

1. Ensure the DB2 CLI/ODBC driver was installed during the DB2 client install.
2. Catalog the DB2 database and node if the database is being accessed from a remote client.

On the Windows platform, you can use the CLI/ODBC Settings GUI to catalog the DB2 database.

3. Optional: Explicitly bind the DB2 CLI/ODBC bind files to the database with the command:

```
db2 bind ~/sqllib/bnd/@db2cli.lst blocking all sqlerror continue \  
messages cli.msg grant public
```

On the Windows platform, you can use the CLI/ODBC Settings GUI to bind the DB2 CLI/ODBC bind files to the database.

4. Optional: Change the DB2 CLI/ODBC configuration keywords by editing the db2cli.ini file, located in the sqllib directory on Windows, and in the sqllib/cfg directory on UNIX platforms.

On the Windows platform, you can use the CLI/ODBC Settings GUI to set the DB2 CLI/ODBC configuration keywords.

Once you have completed the above steps, proceed to setting up your Windows CLI environment, or setting up your UNIX ODBC environment if you are running ODBC applications on UNIX.

Related concepts:

- “Initialization and termination in CLI overview” on page 17

Related tasks:

- “Initializing CLI applications” on page 18
- “Setting up the UNIX ODBC environment” on page 210
- “Setting up the Windows CLI environment” on page 215

Related reference:

- “BIND command” in *Command Reference*
- “CATALOG DATABASE command” in *Command Reference*

Setting up the UNIX ODBC environment

This topic explains how to set up UNIX client access to DB2 for ODBC applications. (If your application is a DB2 CLI application, your CLI environmental setup will be complete once the task in the Prerequisites section is performed.)

Prerequisites:

Before setting up the UNIX ODBC environment, ensure you have set up the CLI environment.

Procedure:

For ODBC applications on UNIX that need to access a DB2 database, follow the steps described below.

1. Ensure that an ODBC driver manager is installed and that each user that will use ODBC has access to it. DB2 does not install an ODBC driver manager, so you must use the ODBC driver manager that was supplied with your ODBC client application or ODBC SDK in order to access DB2 data using that application.
2. Set up `.odbc.ini`, the end-user’s data source configuration. Each user ID has a separate copy of this file in their home directory. Note that the file starts with a dot. Although necessary files are usually updated automatically by the tools on most platforms, users of ODBC on UNIX platforms will have to edit them manually.

Using an ASCII editor, update the file to reflect the appropriate data source configuration information. To register a DB2 database as an ODBC data source there must be one stanza (section) for each DB2 database.

The `.odbc.ini` file must contain the following lines (examples refer to configuration of the SAMPLE database data source):

- in the [ODBC Data Source] stanza:

```
SAMPLE=IBM DB2 ODBC DRIVER
```

which indicates that there is a data source called SAMPLE that uses the IBM DB2 ODBC DRIVER;

- in the [SAMPLE] stanza:

on AIX, for example,

```
[SAMPLE]
Driver=/u/thisuser/sql1lib/lib/libdb2.a
Description=Sample DB2 ODBC Database
```

on the Solaris operating system, for example,

```
[SAMPLE]
Driver=/u/thisuser/sql1lib/lib/libdb2.so
Description=Sample DB2 ODBC Database
```

which indicates that the SAMPLE database is part of the DB2 instance located in the directory /u/thisuser.

With the introduction of the 64-bit development environment, there have been a number of inconsistencies among vendors regarding the interpretation of the sizes of certain parameters. For example, the 64-bit Microsoft ODBC Driver Manager treats SQLHANDLE and SQLLEN as both 64-bits in length, whereas Data Direct Connect and open source ODBC driver managers treat SQLHANDLE as 64-bit, but SQLLEN as 32-bit. The developer must therefore pay careful attention to which version of the DB2 driver is required. Specify the appropriate DB2 driver in the data source stanza, according to the following information:

Table 19. DB2 driver for CLI and ODBC applications

Type of application	DB2 driver to specify
32-bit CLI	libdb2.*
32-bit ODBC Driver Manager	libdb2.*
64-bit CLI	libdb2.*
64-bit ODBC Driver Manager	libdb2o.* (db2o.o for AIX)

Note: The file extension of the DB2 driver to specify depends on the operating system. The extensions are as follows:

- .a - AIX
- .so - Linux, Solaris, HP-IPF
- .sl - HP-PA

3. Ensure that the application execution environment has reference to the ODBC driver manager by including libodbc.a (for AIX) or libodbc.so (for UNIX) in the LIBPATH (for AIX) or LD_LIBRARY_PATH (for UNIX) environment variables.
4. Enable a system-wide .odbc.ini file to be used by setting the ODBCINI environment variable to the fully qualified pathname of the .ini file. Some ODBC driver managers support this feature which allows for centralized control. The following examples show how to set ODBCINI:

in the C shell,

```
setenv ODBCINI /opt/odbc/system_odbc.ini
```

in the Bourne or Korn shell,

```
ODBCINI=/opt/odbc/system_odbc.ini;export ODBCINI
```
5. Once the .odbc.ini file is set up, you can run your ODBC application and access DB2 databases. Refer to the documentation that comes with your ODBC application for additional help and information.

Related concepts:

- “Comparison of DB2 CLI and Microsoft ODBC” on page 9
- “Initialization and termination in CLI overview” on page 17

Related tasks:

- “Building CLI applications on UNIX” on page 219
- “Building CLI routines on UNIX” on page 222
- “Initializing CLI applications” on page 18
- “Setting up the CLI environment” on page 209

Setting up the unixODBC Driver Manager

An ODBC driver manager is not supplied on UNIX platforms as part of the operating system. Using ODBC on UNIX systems requires a separate commercial or open source ODBC driver manager. The unixODBC Driver Manager is an open source ODBC driver manager supported for DB2 ODBC applications on all supported DB2 UNIX platforms. This topic describes how to set up the unixODBC Driver Manager. Please also refer to the unixODBC web site (<http://www.unixodbc.com>), as well as the README files within the unixODBC distribution package for more information.

Support statement:

If you experience problems with the combination of the unixODBC Driver Manager and the DB2 ODBC driver after they have been properly installed and configured, you can contact DB2 Service (<http://www.ibm.com/software/data/db2/udb/support>) for assistance in diagnosing the problem. If the source of the problem lies with the unixODBC Driver Manager, then you can:

- Purchase a service contract for technical support from Easysoft, a commercial sponsor of unixODBC (<http://www.easysoft.com>).
- Participate in any open source support channels at <http://www.unixodbc.com>.

Procedure:

To set up the unixODBC Driver Manager for use with DB2 CLI and ODBC applications:

1. Download the latest unixODBC source code from: <http://www.unixodbc.com>.
2. Untar the source files:

```
gzip -d unixODBC-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar
```
3. For AIX only: configure the C compiler to be thread-enabled:

```
export CC=xlc_r
export CCC=xlc_r
```
4. To compile a 64-bit version of the driver manager using the xlc_r compilers, set the environment variables OBJECT_MODE and CFLAGS:

```
export OBJECT_MODE=64
export CFLAGS=-q64 -DBUILD_REAL_64_BIT_MODE
```
5. Install the driver manager in either your home directory or the default /usr/local prefix:
 - (Home directory) Issue the following command in the directory where you untarred the source files:

```
./configure --prefix=$HOME -DBUILD_REAL_64_BIT_MODE --enable-gui=no
--enable-drivers=no
```
 - (/usr/local as root) Issue the following command:

```
./configure --enable-gui=no --enable-drivers=no
```
6. Optional: Examine all configuration options by issuing the following command:

```
./configure --help
```
7. Build and install the driver manager:

```
make
make install
```

Libraries will be copied to the [prefix]/lib directory, and executables will be copied to the [prefix]/bin directory.

8. Build the application and ensure it is linked to the unixODBC Driver Manager by including the -L[prefix]/lib -lodbc option in the compile and link command.
9. Specify the paths for at least the user INI file (odbc.ini) or the system INI file (odbcinst.ini), and set the ODBC_HOME environment variable to the directory where the system INI file was created.

Important: Provide absolute paths when specifying the paths of the user and system INI files. Do not use relative paths or environment variables.

Related tasks:

- “Setting up the UNIX ODBC environment” on page 210

Related reference:

- “Sample build scripts and configurations for the unixODBC Driver Manager” on page 213

Sample build scripts and configurations for the unixODBC Driver Manager

The unixODBC Driver Manager is an open source ODBC driver manager for use on UNIX platforms. This driver manager is supported for ODBC applications on supported DB2 platforms. This topic presents some examples of possible build scripts and configurations you might want to use when using the unixODBC Driver Manager.

Support statement:

If you experience problems with the combination of the unixODBC Driver Manager and the DB2 ODBC driver after they have been properly installed and configured, you can contact DB2 Service (<http://www.ibm.com/software/data/db2/udb/support>) for assistance in diagnosing the problem. If the source of the problem lies with the unixODBC Driver Manager, then you can:

- Purchase a service contract for technical support from Easysoft, a commercial sponsor of unixODBC (<http://www.easysoft.com>).
- Participate in any open source support channels at <http://www.unixodbc.com>.

Sample build scripts:

The following are sample build scripts for setting up your environment to use the unixODBC Driver Manager.

AIX:

```
#!/bin/sh

echo "Unzipping and extracting"
gzip -d unixODBC-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar

cd unixODBC-2.2.11

#Comment this out if not AIX
```

```

export CC=xlc_r
export CCC=xlc_r

echo "Configuring, compiling and installing"
configure --prefix=$HOME --enable-gui=no --enable-drivers=no
make
make install

echo "Setting ini env vars."
export ODBC_HOME=~/.etc
export ODBC_INI=~/.odbc.ini

#Comment this out if not AIX
echo "Extracting unixODBC libraries"
cd ~/.lib
ar -x libodbc.a
ar -x libodbcinst.a
ar -x libodbcrc.a

echo "\n***Still need to set up your ini files"

```

UNIX (non-AIX):

```

#!/bin/sh

echo "Unzipping and extracting"
gzip -d unixODBC-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar

cd unixODBC-2.2.11

echo "Configuring, compiling and installing"
configure --prefix=$HOME --enable-gui=no --enable-drivers=no
make
make install

echo "Setting ini env vars."
export ODBC_HOME=~/.etc
export ODBC_INI=~/.odbc.ini

echo "\n***Still need to set up your ini files"

```

Sample INI file configurations:

The following are sample user and system INI files for using the unixODBC Driver Manager.

User INI file (odbc.ini):

```

[DEFAULT]
Driver = DB2

[SAMPLE]
DESCRIPTION = Connection to DB2
DRIVER = DB2

```

System INI file (odbcinst.ini):

```

[DEFAULT]
Description = Default Driver
Driver = /u/db2inst1/sqllib/lib/db2.o
fileusage=1
dontdlclose=1

[DB2]
Description = DB2 Driver
Driver = /u/db2inst1/sqllib/lib/db2.o

```



```
fileusage=1
dontdlclose=1

[ODBC]
Trace = yes
Tracefile = /u/user/trc.log
```

This system INI file has the ODBC trace enabled, with the trace log file set to `trc.log`.

Note: If you encounter problems when closing the driver manager (such as during `SQLDisconnect()`), set the value `dontdlclose=1` in the `odbcinst.ini` file, as shown in the example above.

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189

Related tasks:

- “Setting up the unixODBC Driver Manager” on page 212
- “Setting up the UNIX ODBC environment” on page 210

Setting up the Windows CLI environment

This task tells you how to perform Windows client access to DB2 using CLI or ODBC.

Prerequisites:

Before setting up the Windows CLI environment, ensure that the CLI environment has been set up.

Restrictions:

When using the Configuration Assistant on Windows 64-bit platforms, ODBC Data Sources can be configured *only* for 64-bit applications. ODBC Data Sources for 32-bit applications need to be configured using the Microsoft 32-bit ODBC Data Source Administrator (32-bit `odbcad32.exe`) that is included with the Windows 64-bit operating system.

Procedure:

Before DB2 CLI and ODBC applications can successfully access a DB2 database from a Windows client, perform the following steps on the client system:

1. Verify that the Microsoft ODBC Driver Manager and the DB2 CLI/ODBC driver are installed. On Windows operating systems they are both installed with DB2 unless the ODBC component is manually unselected during the install. DB2 will not overwrite a newer version of the Microsoft ODBC Driver Manager if one is found. To verify that they both exist on the machine:
 - a. Start the Microsoft ODBC Data Sources icon in the Control Panel, or run the `odbcad32.exe` command from the command line.
 - b. Click on the “Drivers” tab.

- c. Verify that IBM DB2 ODBC DRIVER - <DB2 Copy Name> is shown in the list. <DB2 Copy name> is the DB2 copy name that you want to use. Refer to Multiple DB2 copies on the same computer (Windows) for more information.

If either the Microsoft ODBC Driver Manager or the IBM DB2 CLI/ODBC driver is not installed, then rerun the DB2 install and select the ODBC component on Windows operating systems.

Note: The latest version of the Microsoft ODBC Driver Manager is included as part of the Microsoft Data Access Components (MDAC) and is available for download from <http://www.microsoft.com/data/>.

2. Register the DB2 database with the ODBC driver manager as a data source. On Windows operating systems you can make the data source available to all users of the system (a system data source), or only the current user (a user data source). Use either of these methods to add the data source:

- Using the Configuration Assistant:
 - a. Select the DB2 database alias that you want to add as a data source.
 - b. Click on the "Properties" push button. The Database Properties window opens.
 - c. Select the "Register this database for ODBC" check box.
 - d. Use the radio buttons to add the data source as either a user, system, or file data source.
- Using the Microsoft ODBC Administration tool, which you can access from the icon in the Control Panel or by running `odbcad32.exe` from the command line:
 - a. The list of user data sources appears by default. If you want to add a system data source click on the "System DSN" button, or the "System DSN" tab (depending on the platform).
 - b. Click on the "Add" push button.
 - c. Double-click on the IBM DB2 ODBC DRIVER - <DB2 Copy name> in the list. <DB2 Copy name> is the DB2 copy name that you want to use.
 - d. Select the DB2 database to add and click on OK.
- Use the CATALOG command to register the DB2 database with the ODBC driver manager as a data source:

```
CATALOG [ user | system ] ODBC DATA SOURCE
```

Using this command, an administrator could create a command line processor script to register the required databases. This script could then be run on all machines that require access to DB2 databases through ODBC.

3. Optional: Configure the DB2 CLI/ODBC driver using the Configuration Assistant:
 - a. Select the DB2 database alias you want to configure.
 - b. Click on the "Properties" push button. The Database Properties window opens.
 - c. Click on the "Settings" push button. The CLI/ODBC Settings window opens.
 - d. Click on the "Advanced" push button. You can set the configuration keywords in the window that opens. These keywords are associated with the database alias name, and affect all DB2 CLI/ODBC applications that access the database.

4. If you have installed ODBC access (as described above), you can now access DB2 data using ODBC applications.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Initialization and termination in CLI overview” on page 17
- “Multiple DB2 copies on the same computer (Windows)” in *Administration Guide: Implementation*

Related tasks:

- “Building CLI applications on Windows” on page 228
- “Building CLI routines on Windows” on page 231
- “Initializing CLI applications” on page 18
- “Setting up the CLI environment” on page 209

Selecting a different DB2 copy for your Windows CLI application

By default, DB2 CLI applications running on Windows systems make use of the default DB2 copy. However, applications can use any DB2 copy that is installed on the system.

Prerequisites:

Ensure your Windows DB2 CLI environment is set up.

Restrictions:

See *Multiple DB2 copies on the same computer (Windows)* and *Client connectivity using multiple DB2 copies (Windows)*

Procedure:

The following are different methods that allow DB2 CLI applications to successfully access a different DB2 copy on the Windows platform.

- Using the DB2 command window from the Start → Programs → IBM DB2 → <DB2 Copy Name> → Command Line Tools → DB2 Command Window: the command window is already set up with the correct environment variables for the particular DB2 copy chosen.
- Using db2envvar.bat from a command window:
 1. Open a command window.
 2. Run the db2envvar.bat file using the fully qualified path for the DB2 copy that you want the application to use:

```
<DB2 Copy install dir>\bin\db2envvar.bat
```
 3. Run the DB2 CLI application from the same command window.

This will set up all the environment variables for the selected DB2 copy in the command window where the db2envvar.bat was run. Once the command window has been closed and a new one opened, the DB2 CLI application will run against the default DB2 Copy unless the db2envvar.bat for another DB2 copy is run again.
- Using the **db2SelectDB2Copy** API: For applications that are dynamically linked, you can call this API prior to loading any DB2 DLLs within your application process. This API sets up the required environment for your application to use

the DB2 copy that you want to use. The /delayload linking option can be used to delay the loading of any DB2 DLL. For example, if your DB2 CLI application links db2api.lib, then you must use the /delayload option of your linker to delay the load db2app.dll:

```
c1 -Zi -MDd -Tp App.C /link /DELAY:nobind /DELAYLOAD:db2app.dll  
advapi32.lib psapi.lib db2api.lib delayimp.lib
```

To use the API, you will need to include db2ApiInstall.h, which will force your application to statically link in db2ApiInstall.lib.

- Using LoadLibraryEx: Instead of using LoadLibrary, you can call LoadLibraryEx with the LOAD_WITH_ALTERED_SEARCH_PATH parameter to load the db2app.dll that corresponds to the version of the DB2 copy you want to use.

For example:

```
HMODULE hLib = LoadLibraryEx("c:\\sql1ib\\bin\\db2app.dll",  
NULL, LOAD_WITH_ALTERED_SEARCH_PATH);
```

Related concepts:

- “Multiple DB2 copies on the same computer (Windows)” in *Administration Guide: Implementation*

Related tasks:

- “Setting up the Windows CLI environment” on page 215
- “Client connectivity using multiple DB2 copies (Windows)” in *Administration Guide: Implementation*
- “Setting environment variables on Windows” in *Administration Guide: Implementation*
- “Setting the current instance environment variables” in *Administration Guide: Implementation*

Related reference:

- “db2SelectDB2Copy API - Select the DB2 copy to be used by your application” in *Administrative API Reference*
- “Multiple DB2 copies roadmap” in *Administration Guide: Implementation*

Chapter 25. Building CLI applications

UNIX	219	Linux	227
Building CLI applications on UNIX	219	Build script for Linux applications	227
Building CLI multi-connection applications on UNIX	220	Build script for Linux routines.	227
Building CLI routines on UNIX	222	Solaris.	228
AIX	224	Build script for Solaris applications	228
Build script for AIX applications	224	Build script for Solaris routines	228
CLI applications and configuration files on AIX	224	Windows.	228
Build script for AIX routines	225	Building CLI applications on Windows.	228
CLI routines and configuration files on AIX	226	Building CLI multi-connection applications on Windows.	229
HP-UX	227	Building CLI routines on Windows	231
Build script for HP-UX applications	227	Batch file for Windows applications	233
Build script for HP-UX routines	227	Batch file for Windows routines	233

UNIX

The following sections describe how to build CLI applications and routines on supported UNIX operating systems. They also show sample build scripts for each operating system and descriptions of the compile and link options used within the build scripts.

Building CLI applications on UNIX

DB2 provides build scripts for compiling and linking CLI programs. These are located in the `sqllib/samples/cli` directory, along with sample programs that can be built with these files.

The script file `bldapp` contains the commands to build a DB2 CLI application. It takes up to four parameters, represented inside the script file by the variables `$1`, `$2`, `$3`, and `$4`.

The parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for CLI applications that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: the second parameter, `$2`, specifies the name of the database to which you want to connect; the third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password.

If the program contains embedded SQL, indicated by the `.sql` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

Procedure:

The following examples show you how to build and run CLI applications.

To build the sample program `tbinfo` from the source file `tbinfo.c`, enter:

```
bldapp tbinfo
```

The result is an executable file, `tbinfo`. You can run the executable file by entering the executable name:

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:

```
bldapp dbusemx
```

2. If connecting to another database on the same instance, also enter the database name:

```
bldapp dbusemx database
```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

```
bldapp dbusemx database userid password
```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

```
dbusemx
```

2. If accessing another database on the same instance, enter the executable name and the database name:

```
dbusemx database
```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

```
dbusemx database userid password
```

Related tasks:

- “Building CLI routines on UNIX” on page 222
- “Configuring the database application development environment” in *Getting Started with Database Application Development*
- “Setting up the UNIX application development environment” in *Getting Started with Database Application Development*

Related samples:

- “`bldapp -- Builds AIX CLI applications`”
- “`dbusemx.sqc -- How to execute embedded SQL statements in CLI`”
- “`tbinfo.c -- How to get information about tables from the system catalog tables`”

Building CLI multi-connection applications on UNIX

DB2 provides build scripts for compiling and linking CLI programs. These are located in the `sqllib/samples/cli` directory, along with sample programs that can be built with these files.

The build file, `bldmc`, contains the commands to build a DB2 multi-connection program, requiring two databases. The compile and link options are the same as those used in `bldapp`.

The first parameter, \$1, specifies the name of your source file. The second parameter, \$2, specifies the name of the first database to which you want to connect. The third parameter, \$3, specifies the second database to which you want to connect. These are all required parameters.

Note: The makefile hardcodes default values of "sample" and "sample2" for the database names (\$2 and \$3, respectively) so if you are using the makefile, and accept these defaults, you only have to specify the program name (the \$1 parameter). If you are using the bldmc script, you must specify all three parameters.

Optional parameters are not required for a local connection, but are required for connecting to a server from a remote client. These are: \$4 and \$5 to specify the user ID and password, respectively, for the first database; and \$6 and \$7 to specify the user ID and password, respectively, for the second database.

Procedure:

For the multi-connection sample program, dbmconx, you require two databases. If the sample database is not yet created, you can create it by entering db2samp1 on the command line. The second database, here called sample2, can be created with one of the following commands:

If creating the database locally:

```
db2 create db sample2
```

If creating the database remotely:

```
db2 attach to <node_name>
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node <node_name>
```

where <node_name> is the database partition where the database resides.

Multi-connection also requires that the TCP/IP listener is running. To ensure it is, do the following:

1. Set the environment variable DB2COMM to TCP/IP as follows:

```
db2set DB2COMM=TCPIP
```
2. Update the database manager configuration file with the TCP/IP service name as specified in the services file:

```
db2 update dbm cfg using SVCENAME <TCP/IP service name>
```

Each instance has a TCP/IP service name listed in the services file. Ask your system administrator if you cannot locate it or do not have the file permission to read the services file. On UNIX and Linux systems, the services file is located in: /etc/services

3. Stop and restart the database manager in order for these changes to take effect:

```
db2stop
db2start
```

The dbmconx program consists of five files:

dbmconx.c

Main source file for connecting to both databases.

dbmconx1.sqc

Source file for creating a package bound to the first database.

dbmconx1.h

Header file for dbmconx1.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the first database.

dbmconx2.sqc

Source file for creating a package bound to the second database.

dbmconx2.h

Header file for dbmconx2.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the second database.

To build the multi-connection sample program, dbmconx, enter:

```
bldmc dbmconx sample sample2
```

The result is an executable file, dbmconx.

To run the executable file, enter the executable name:

```
dbmconx
```

The program demonstrates a two-phase commit to two databases.

Related concepts:

- “Multisite updates (two phase commit) in CLI applications” on page 137

Related reference:

- “svcname - TCP/IP service name configuration parameter” in *Performance Guide*

Related samples:

- “bldmc -- Builds AIX CLI multi-connection applications”
- “bldmc -- Builds HP-UX CLI multi-connection applications”
- “bldmc -- Builds Linux CLI multi-connection applications”
- “bldmc -- Builds Solaris CLI multi-connection applications”
- “dbmconx.c -- How to use multiple databases with embedded SQL.”
- “dbmconx1.h -- Functions used in dbmconx.c”
- “dbmconx1.sqc -- This file contains functions used in dbmconx.c”
- “dbmconx2.h -- Functions used in dbmconx.c”
- “dbmconx2.sqc -- This file contains functions used in dbmconx.c”

Building CLI routines on UNIX

DB2 provides build scripts for compiling and linking CLI programs. These are located in the `sqllib/samples/cli` directory, along with sample programs that can be built with these files.

The script file `bldrtn` contains the commands to build DB2 CLI routines (stored procedures and user-defined functions). `bldrtn` creates a shared library on the server. It takes a parameter for the source file name, represented inside the script file by the variable `$1`.

Procedure:

To build the sample program `spserver` from the source file `spserver.c`:

1. Enter the build script name and program name:

```
bldrtn spserver
```

The script file copies the shared library to the `sqllib/function` directory.

2. Next, catalog the routines by running the `spcat` script on the server:

```
spcat
```

This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by calling `spcreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spcreate.db2` scripts individually.

3. Then, unless this is the first time the shared library was built, stop and restart the database to allow the new version of the shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the CLI client application, `spclient`, that calls the routines within the shared library.

The client application can be built like any other CLI client application by using the script file, `bldapp`.

To access the shared library, run the sample client application by entering:

```
spclient database userid password
```

where

database

Is the name of the database to which you want to connect. The name could be `sample`, or its alias, or another database name.

userid Is a valid user ID.

password

Is a valid password.

The client application accesses the shared library, `spserver`, and executes the routines on the server database. The output is returned to the client application.

Related tasks:

- “Setting up the UNIX application development environment” in *Getting Started with Database Application Development*
- “Building CLI applications on UNIX” on page 219

Related samples:

- “`bldrtn` -- Builds AIX CLI routines (stored procedures and UDFs)”
- “`spclient.c` -- Call various stored procedures”
- “`spserver.c` -- Definition of various types of stored procedures”
- “`spcat` -- To catalog stored procedures on UNIX (C)”
- “`spcreate.db2` -- How to catalog the stored procedures contained in `spserver.sqc` (C)”

- “spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)”

AIX

Build script for AIX applications

The script for building CLI applications on AIX is called `bldapp` and is located in the `sqliib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI applications with the AIX IBM C compiler, are demonstrated in the `sqliib/samples/cli/bldapp` build script also.

CLI applications and configuration files on AIX

The following section describes how to build CLI applications with configuration files. This method of building CLI applications is only available on AIX using the VisualAge C++ compiler.

Building CLI applications with configuration files: The configuration file, `cli.icc`, in `sqliib/samples/cli` allows you to build DB2 CLI programs.

Procedure:

To use the configuration file to build the DB2 CLI sample program `tbinfo` from the source file `tbinfo.c`, do the following:

1. Set the CLI environment variable:

```
export CLI=tbinfo
```

2. If you have a `cli.ics` file in your working directory, produced by building a different program with the `cli.icc` file, delete the `cli.ics` file with this command:

```
rm cli.ics
```

An existing `cli.ics` file produced for the same program you are going to build again does not have to be deleted.

3. Compile the sample program by entering:

```
vacbld cli.icc
```

Note: The `vacbld` command is provided by VisualAge® C++.

The result is an executable file, `tbinfo`. You can run the program by entering the executable name:

```
tbinfo
```

Building and running embedded SQL applications

You use the configuration file after the program is precompiled with the `embprep` file. The `embprep` file precompiles the source file and binds the program to the database. You use the `cli.icc` configuration file to compile the precompiled file.

There are three ways to precompile the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:

```
embprep dbusemx
```

2. If connecting to another database on the same instance, also enter the database name:

```
embprep dbusemx database
```

3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

```
embprep dbusemx database userid password
```

The result is a precompiled C file, `dbusemx.c`.

After it is precompiled, the C file can be compiled with the `cli.icc` file as follows:

1. Set the CLI environment variable to the program name by entering:

```
export CLI=dbusemx
```

2. If you have a `cli.ics` file in your working directory, produced by building a different program with the `cli.icc` or `cliapi.icc` file, delete the `cli.ics` file with this command:

```
rm cli.ics
```

An existing `cli.ics` file produced for the same program you are going to build again does not have to be deleted.

3. Compile the sample program by entering:

```
vacbld cli.icc
```

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

```
dbusemx
```

2. If accessing another database on the same instance, enter the executable name and the database name:

```
dbusemx database
```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

```
dbusemx database userid password
```

Related tasks:

- “Building embedded SQL and DB2 API applications in C or C++ with configuration files (AIX)” in *Developing Embedded SQL Applications*
- “Building embedded SQL stored procedures in C or C++ with configuration files” in *Developing Embedded SQL Applications*
- “Building user-defined functions in C or C++ with configuration files (AIX)” in *Developing Embedded SQL Applications*
- “Building CLI stored procedures with configuration files” on page 226
- “Building embedded SQL applications written in VisualAge C++ with configuration files” in *Developing Embedded SQL Applications*

Related samples:

- “`dbusemx.sqc` -- How to execute embedded SQL statements in CLI”
- “`tbinfo.c` -- How to get information about tables from the system catalog tables”

Build script for AIX routines

The script for building CLI routines (stored procedures and user-defined functions) on AIX is called `bldrtn` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI routines with the AIX IBM C compiler, are demonstrated in the `sqllib/samples/cli/bldrtn` build script also.

CLI routines and configuration files on AIX

The following section describes how to build CLI routines with configuration files. This method of building CLI routines is only available on AIX using the VisualAge C++ compiler.

Building CLI stored procedures with configuration files: The configuration file, `clis.icc`, in `sqllib/samples/cli`, allows you to build DB2 CLI stored procedures.

Procedure:

To use the configuration file to build the DB2 CLI stored procedure `spserver` from the source file `spserver.c`, do the following:

1. Set the CLIS environment variable to the program name by entering:

```
export CLIS=spserver
```
2. If you have a `clis.ics` file in your working directory, produced by building a different program with the `clis.icc` file, delete the `clis.ics` file with this command:

```
rm clis.ics
```

An existing `clis.ics` file produced for the same program you are going to build again does not have to be deleted.

3. Compile the sample program by entering:

```
vacbld clis.icc
```

Note: The `vacbld` command is provided by VisualAge C++.

The stored procedure is copied to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `screate.db2` script on the server. First, connect to the database with the user ID and password of the instance where the database is located:

```
db2 connect to sample userid password
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf screate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure `spserver`, you can build the CLI client application `spclient` that calls the stored procedure. You can build `spclient` by using the configuration file, `cli.icc`.

To call the stored procedure, run the sample client application by entering:

```
spclient database userid password
```

where

database

Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

userid Is a valid user ID.

password

Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

Related tasks:

- “Building CLI routines on UNIX” on page 222

Related samples:

- “`spclient.c` -- Call various stored procedures”
- “`spcreate.db2` -- How to catalog the stored procedures contained in `spserver.sqc` (C)”
- “`spdrop.db2` -- How to uncatalog the stored procedures contained in `spserver.sqc` (C)”

HP-UX

Build script for HP-UX applications

The script for building CLI applications on HP-UX is called `bldapp` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI applications with the HP-UX C compiler, are demonstrated in the `sqllib/samples/cli/bldapp` build script also.

Build script for HP-UX routines

The script for building CLI routines (stored procedures and user-defined functions) on HP-UX is called `bldrtn` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI routines with the HP-UX C compiler, are demonstrated in the `sqllib/samples/cli/bldrtn` build script also.

Linux

Build script for Linux applications

The script for building CLI applications on Linux is called `bldapp` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI applications with the GNU/Linux `gcc` compiler, are demonstrated in the `sqllib/samples/cli/bldapp` build script also.

Build script for Linux routines

The script for building CLI routines (stored procedures and user-defined functions) on Linux is called `bldrtn` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI routines with the GNU/Linux `gcc` compiler, are demonstrated in the `sqllib/samples/cli/bldrtn` build script also.

Solaris

Build script for Solaris applications

The script for building CLI applications on Solaris is called `bldapp` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI applications with the Solaris C compiler, are demonstrated in the `sqllib/samples/cli/bldapp` build script also.

Build script for Solaris routines

The script for building CLI routines (stored procedures and user-defined functions) on Solaris is called `bldrtn` and is located in the `sqllib/samples/cli` directory. The compile and link options recommended by DB2 for building CLI routines with the Solaris C compiler, are demonstrated in the `sqllib/samples/cli/bldrtn` build script also.

Windows

The following sections describe how to build CLI applications and routines on supported Windows operating systems. They also provide sample batch files for building DB2 programs as well as descriptions of the compile and link options used in the batch files.

Building CLI applications on Windows

DB2 provides batch files for compiling and linking CLI programs. These are located in the `sqllib\samples\cli` directory, along with sample programs that can be built with these files.

The batch file `bldapp.bat` contains the commands to build a DB2 CLI program. It takes up to four parameters, represented inside the batch file by the variables `%1`, `%2`, `%3`, and `%4`.

The parameter, `%1`, specifies the name of your source file. This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: the second parameter, `%2`, specifies the name of the database to which you want to connect; the third parameter, `%3`, specifies the user ID for the database, and `%4` specifies the password.

If the program contains embedded SQL, indicated by the `.sql` or `.sqlx` extension, then the `embprep.bat` batch file is called to precompile the program, producing a program file with either a `.c` or a `.cxx` extension, respectively.

Procedure:

The following examples show you how to build and run CLI applications.

To build the sample program `tbinfo` from the source file `tbinfo.c`, enter:

```
bldapp tbinfo
```

The result is an executable file `tbinfo`. You can run the executable file by entering the executable name:

```
tbinfo
```

Building and running embedded SQL applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:

```
bldapp dbusemx
```
2. If connecting to another database on the same instance, also enter the database name:

```
bldapp dbusemx database
```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:

```
bldapp dbusemx database userid password
```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the sample database on the same instance, simply enter the executable name:

```
dbusemx
```
2. If accessing another database on the same instance, enter the executable name and the database name:

```
dbusemx database
```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:

```
dbusemx database userid password
```

Related tasks:

- “Building CLI routines on Windows” on page 231
- “Setting up the CLI environment” on page 209
- “Setting up the Windows CLI environment” on page 215

Related samples:

- “`bldapp.bat` -- Builds C applications on Windows”
- “`embprep.bat` -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows”
- “`dbusemx.sqc` -- How to execute embedded SQL statements in CLI”
- “`tbinfo.c` -- How to get information about tables from the system catalog tables”

Building CLI multi-connection applications on Windows

DB2 provides batch files for compiling and linking CLI programs. These are located in the `sqllib\samples\cli` directory, along with sample programs that can be built with these files.

The batch file, `bldmc.bat`, contains the commands to build a DB2 multi-connection program requiring two databases. The compile and link options are the same as those used in `bldapp.bat`.

The first parameter, `%1`, specifies the name of your source file. The second parameter, `%2`, specifies the name of the first database to which you want to

connect. The third parameter, %3, specifies the second database to which you want to connect. These are all required parameters.

Note: The makefile hardcodes default values of "sample" and "sample2" for the database names (%2 and %3, respectively) so if you are using the makefile, and accept these defaults, you only have to specify the program name (the %1 parameter). If you are using the bldmc.bat file, you must specify all three parameters.

Optional parameters are not required for a local connection, but are required for connecting to a server from a remote client. These are: %4 and %5 to specify the user ID and password, respectively, for the first database; and %6 and %7 to specify the user ID and password, respectively, for the second database.

Procedure:

For the multi-connection sample program, dbmconx, you require two databases. If the sample database is not yet created, you can create it by entering db2samp1 on the command line. The second database, here called sample2, can be created with one of the following commands:

If creating the database locally:

```
db2 create db sample2
```

If creating the database remotely:

```
db2 attach to <node_name>
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node <node_name>
```

where <node_name> is the database partition where the database resides.

Multi-connection also requires that the TCP/IP listener is running. To ensure it is, do the following:

1. Set the environment variable DB2COMM to TCP/IP as follows:

```
db2set DB2COMM=TCPIP
```
2. Update the database manager configuration file with the TCP/IP service name as specified in the services file:

```
db2 update dbm cfg using SVCENAME <TCP/IP service name>
```

Each instance has a TCP/IP service name listed in the services file. Ask your system administrator if you cannot locate it or do not have the file permission to read the services file.

3. Stop and restart the database manager in order for these changes to take effect:

```
db2stop
db2start
```

The dbmconx program consists of five files:

dbmconx.c

Main source file for connecting to both databases.

dbmconx1.sqc

Source file for creating a package bound to the first database.

dbmconx1.h

Header file for dbmconx1.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the first database.

dbmconx2.sqc

Source file for creating a package bound to the second database.

dbmconx2.h

Header file for dbmconx2.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the second database.

To build the multi-connection sample program, dbmconx, enter:

```
bldmc dbmconx sample sample2
```

The result is an executable file, dbmconx.

To run the executable file, enter the executable name:

```
dbmconx
```

The program demonstrates a two-phase commit to two databases.

Related concepts:

- “Multisite updates (two phase commit) in CLI applications” on page 137

Related reference:

- “svcename - TCP/IP service name configuration parameter” in *Performance Guide*

Related samples:

- “bldmc.bat -- Builds Windows CLI multi-connection applications”
- “dbmconx.c -- How to use multiple databases with embedded SQL.”
- “dbmconx1.h -- Functions used in dbmconx.c”
- “dbmconx1.sqc -- This file contains functions used in dbmconx.c”
- “dbmconx2.h -- Functions used in dbmconx.c”
- “dbmconx2.sqc -- This file contains functions used in dbmconx.c”

Building CLI routines on Windows

DB2 provides batch files for compiling and linking CLI programs. These are located in the sql11ib\samples\cli directory, along with sample programs that can be built with these files.

The batch file bldrtn.bat contains the commands to build CLI routines (stored procedures and user-defined functions). bldrtn.bat creates a DLL on the server. It takes one parameter, represented inside the batch file by the variable %1, which specifies the name of your source file. The batch file uses the source file name for the DLL name.

Procedure:

To build the spserver DLL from the source file spserver.c:

1. Enter the batch file name and program name:

```
bldrtn spserver
```

The batch file uses the module definition file `spserver.def`, contained in the same directory as the CLI sample programs, to build the DLL. The batch file then copies the DLL, `spserver.dll`, to the server in the path `sqllib\function`.

2. Next, catalog the routines by running the `spcat` script on the server:

```
spcat
```

This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by calling `spscreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spscreate.db2` scripts individually.

3. Then, unless this is the first time the shared library was built, stop and restart the database to allow the new version of the shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the DLL `spserver`, you can build the CLI client application `spclient` that calls the routines within it.

You can build `spclient` by using the script file, `bldapp`.

To call the routines, run the sample client application by entering:

```
spclient database userid password
```

where

database

Is the name of the database to which you want to connect. The name could be `sample`, or its alias, or another database name.

userid Is a valid user ID.

password

Is a valid password.

The client application accesses the DLL, `spserver`, which executes the routines on the server database. The output is returned to the client application.

Related tasks:

- “Building CLI applications on Windows” on page 228
- “Setting up the CLI environment” on page 209
- “Setting up the Windows CLI environment” on page 215

Related samples:

- “`bldapp.bat` -- Builds C applications on Windows”
- “`bldrtn.bat` -- Builds C routines (stored procedures and UDFs) on Windows”
- “`spcat` -- To catalog stored procedures on UNIX (C)”
- “`spscreate.db2` -- How to catalog the stored procedures contained in `spserver.sqc` (C)”
- “`spdrop.db2` -- How to uncatalog the stored procedures contained in `spserver.sqc` (C)”
- “`spclient.c` -- Call various stored procedures”
- “`spserver.c` -- Definition of various types of stored procedures”

Batch file for Windows applications

The batch file for building CLI applications on Windows is called `bldapp.bat` and is located in the `sql11b/samples/cli` directory. The compile and link options recommended by DB2 for building CLI applications with the Microsoft Visual C++ compiler, are demonstrated in the `sql11b/samples/cli/bldapp` build script also.

Batch file for Windows routines

The batch file for building CLI routines (stored procedures and user-defined functions) on Windows is called `bldrtn.bat` and is located in the `sql11b/samples/cli` directory. The compile and link options recommended by DB2 for building CLI routines with the Microsoft Visual C++ compiler, are demonstrated in the `sql11b/samples/cli/bldrtn` build script also.

Chapter 26. CLI sample programs

CLI sample programs

DB2 CLI includes various sample applications located in the following location:

- On Windows operating systems: %DB2PATH%\sqllib\samples\cli (where %DB2PATH% is a variable that determines where DB2 is installed)
- On UNIX: \$HOME/sqllib/samples/cli (where \$HOME is the home directory of the instance owner)

The README file in the same directory lists each sample along with an explanation, and describes how to build the samples using the makefile and build files provided.

Related tasks:

- “Building CLI applications on UNIX” on page 219
- “Building CLI applications on Windows” on page 228
- “Building CLI routines on UNIX” on page 222
- “Building CLI routines on Windows” on page 231
- “Setting up the CLI environment” on page 209

Related reference:

- “CLI samples” on page 235

CLI samples

UNIX directory: sqllib/samples/cli. Windows directory: sqllib\samples\cli.

Table 20. Sample CLI program files

Type of sample	Sample program name	Program description
ADMIN_CMD samples - Programs that demonstrate the use of ADMIN_CMD stored procedure for performing administration tasks	admincmd_autoconfigure.c	How to autoconfigure a database using ADMIN_CMD routine
	admincmd_contacts.c	How to add, update and drop contacts and contactgroups using ADMIN_CMD routine
	admincmd_export.c	How to export data using ADMIN_CMD routine.
	admincmd_import.c	How to import data using ADMIN_CMD routine.
	admincmd_describe.c	How to describe table and indices using ADMIN_CMD routine.
	admincmd_quiesce.c	How to quiesce tablespaces and database using ADMIN_CMD routine.
	admincmd_onlinebackup.c	How to perform online backup using ADMIN_CMD routine.
	admincmd_updateconfig.c	How to update and reset the Database configuration and Database Manager Configuration Parameters using ADMIN_CMD routine.

Table 20. Sample CLI program files (continued)

Type of sample	Sample program name	Program description
Installation image level - Samples that deal with the DB2 and CLI installation image level.	ilinfo.c	How to get and set installation level information (such as the version of the CLI driver).
	cli_info.c	How to get and set client level information.
Client level - Samples that deal with the client level.	clihandl.c	How to allocate and free handles.
	clisqlca.c	How to work with SQLCA data.
	getdbmcfparams.c	How to get Database Manager Configuration parameters.
Instance level - Samples that deal with the instance level.	ininfo.c	How to get and set instance level information.
	dbcongui.c	How to connect to a database with a Graphical User Interface (GUI).
Database level - Samples that deal with database objects.	dbconn.c	How to connect and disconnect from a database.
	dbinfo.c	How to get and set information at a database level.
	dbmcon.c	How to connect and disconnect from multiple databases.
	dbmconx.c	How to connect and disconnect from multiple databases with embedded SQL.
	dbmconx1.h	Header file for dbmconx1.sqc.
	dbmconx1.sqc	Embedded SQL file for the dbmconx program.
	dbmconx2.h	Header file for dbmconx2.sqc.
	dbmconx2.sqc	Embedded SQL file for the dbmconx program.
	dbnative.c	How to translate a statement that contains an ODBC escape clause to a data source specific format.
	dbuse.c	How to use database objects.
	dbusemx.sqc	How to use database objects with embedded SQL.
	dbxamon.c	How to show and roll back indoubt transactions.
	getdbcfgparams.c	How to get Database Configuration parameters.

Table 20. Sample CLI program files (continued)

Type of sample	Sample program name	Program description
Table level - Samples that deal with table objects.	getmessage.c	How to get error message in the required locale with token replacement.
	tbast.c	How to use staging table for updating deferred AST.
	tbcompress.c	How to create tables with null and default value compression option.
	tbconstr.c	How to work with table constraints.
	tbcreate.c	How to create, alter, and drop tables.
	tbinfo.c	How to get and set information at a table level.
	tbload.c	How to insert data using the CLI LOAD utility.
	tbmod.c	How to modify information in a table.
	tbonlineinx.c	How to create and reorg indexes on a table.
	tbread.c	How to read information in a table.
	tbrunstats.c	How to perform runstats on a table.
	tbtemp.c	How to use a declared temporary table.
	tbumqt.c	How to use user materialized query tables (summary tables).
Data type level - Samples that deal with data types.	dtinfo.c	How to get information about data types.
	dtlob.c	How to read and write LOB data.
	dtudt.c	How to create, use, and drop user defined distinct types.
Stored procedure level - Samples that demonstrate stored procedures.	spcat	Stored procedure catalog script for the spserver program. This script calls spdrop.db2 and spcreate.db2.
	spcreate.db2	CLP script to issue CREATE PROCEDURE statements.
	spdrop.db2	CLP script to drop stored procedures from the catalog.
	spclient.c	Client program used to call the server functions declared in spserver.c.
	spserver.c	Stored procedure functions built and run on the server.
	spcli res.c	Client application that demonstrates the difference between SQLMoreResults and SQLNextResults for multiple result sets.
	spcall.c	Client program for calling any stored procedure.
UDF level - Samples that demonstrate user defined functions.	udfcli.c	Client application which calls the user defined function in udfsrv.c.
	udfsrv.c	User defined function ScalarUDF called by udfcli.c.

Table 20. Sample CLI program files (continued)

Type of sample	Sample program name	Program description
Common utility files	utilcli.c	Utility functions used in CLI samples.
	utilcli.h	Header file for utility functions used in CLI samples.

Related concepts:

- “Building embedded SQL applications using the sample build script” in *Developing Embedded SQL Applications*
- “Error-checking utilities” in *Developing Embedded SQL Applications*
- “Sample files” in *Samples Topics*

Part 4. IBM DB2 Driver for ODBC and CLI

Chapter 27. IBM DB2 Driver for ODBC and CLI overview

The IBM DB2 Driver for ODBC and CLI provides runtime support for the DB2 CLI application programming interface (API) and the ODBC API. Though the DB2 Client and DB2 Runtime Client both support the DB2 CLI and ODBC APIs, this driver is not a part of either DB2 client. It is available separately, installed separately, and supports a subset of the functionality of the DB2 clients.

Advantages of the IBM DB2 Driver for ODBC and CLI:

- The driver has a much smaller footprint than the DB2 Client and the DB2 Runtime Client.
- You can have multiple installations of the driver on a single machine.
- You can install the driver on a machine that already has a DB2 client installed.
- You can include the driver in your database application installation package, and redistribute the driver with your applications. Under certain conditions, you can redistribute the the driver with your database applications royalty-free.
- The driver can reside on an NFS mounted filesystem.

Functionality of the IBM DB2 Driver for ODBC and CLI:

The IBM DB2 Driver for ODBC and CLI provides:

- runtime support for the DB2 CLI API;
- runtime support for the ODBC API;
- runtime support for the XA API;
- database connectivity;
- LDAP Database Directory support; and
- tracing, logging, and diagnostic support.

See: Restrictions of the IBM DB2 Driver for ODBC and CLI.

Related concepts:

- “Introduction to DB2 CLI and ODBC” on page 3

Related tasks:

- “Obtaining the IBM DB2 Driver for ODBC and CLI” on page 243
- “Configuring the IBM DB2 Driver for ODBC and CLI” on page 249
- “Connecting to databases with the IBM DB2 Driver for ODBC and CLI” on page 257
- “Deploying the IBM DB2 Driver for ODBC and CLI with database applications” on page 269
- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI” on page 263
- “Uninstalling the IBM DB2 Driver for ODBC and CLI” on page 273

Chapter 28. Obtaining the IBM DB2 Driver for ODBC and CLI

Obtaining the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It is available to download from the internet, and it is on the DB2 Version 9 install CD.

Prerequisites:

To obtain the IBM DB2 Driver for ODBC and CLI, you need:

- internet access, so you can download the driver; or
- a DB2 Version 9 installation CD.

Procedure:

You can obtain the IBM DB2 Driver for ODBC and CLI by doing either of the following:

- Download the driver from: <http://www-306.ibm.com/software/data/db2/ad>.

or

- Copy the driver from the DB2 Version 9 install CD.

The driver is in a compressed file called “db2_driver_for_odbc_cli.zip” on Windows operating systems, and “db2_driver_for_odbc_cli.tar.Z” on other operating systems.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Chapter 29. Installing the IBM DB2 Driver for ODBC and CLI

Installing the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed separately.

Prerequisites:

To install the IBM DB2 Driver for ODBC and CLI, you need:

- to obtain the compressed file that contains the driver.
See: Obtaining the IBM DB2 Driver for ODBC and CLI.

Procedure:

There is no installation program for the IBM DB2 Driver for ODBC and CLI. You must install the driver manually:

1. Copy the compressed file that contains the driver onto the target machine from the internet or a DB2 Version 9 installation CD.
2. Uncompress that file into your chosen install directory on the target machine.
3. Optional: remove the compressed file.

Example:

If you are installing the IBM DB2 Driver for ODBC and CLI under the following conditions:

- the operating systems on the target machine is AIX; and
- the DB2 Version 9 CD is mounted on the target machine.

the steps you would follow are:

1. Create the directory `$HOME/db2_cli_odbc_driver`, where you will install the driver.
2. Locate the compressed file that contains the driver on the install CD. In this scenario, the file would be called `db2_driver_for_odbc_cli.tar.Z`.
3. Copy `db2_driver_for_odbc_cli.tar.Z` to the install directory, `$HOME/db2_cli_odbc_driver`.
4. Uncompress `db2_driver_for_odbc_cli.tar.Z`:

```
cd $HOME/db2_cli_odbc_driver
uncompress db2_driver_for_odbc_cli.tar.Z
tar -xvf db2_driver_for_odbc_cli.tar
```
5. Delete `db2_driver_for_odbc_cli.tar.Z`.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Obtaining the IBM DB2 Driver for ODBC and CLI” on page 243
- “Installing multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine” on page 246

- “Installing the IBM DB2 Driver for ODBC and CLI on a machine with an existing DB2 client” on page 247
- “Uninstalling the IBM DB2 Driver for ODBC and CLI” on page 273

Installing multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed separately.

You can install multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine. You might want to do this if you have two database applications on the same machine that require different versions of the driver.

Prerequisites:

To install multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine, you need:

- to obtain the compressed file that contains the driver.
See: Obtaining the IBM DB2 Driver for ODBC and CLI.

Procedure:

For each copy of the IBM DB2 Driver for ODBC and CLI that you are installing:

1. create a unique target installation directory; and
2. follow the installation steps outlined in Installing the IBM DB2 Driver for ODBC and CLI.

Example:

If you are installing two copies of the IBM DB2 Driver for ODBC and CLI under the following conditions:

- the operating systems on the target machine is AIX; and
- the DB2 Version 9 CD is mounted on the target machine.

the steps you would follow are:

1. Create the two directories, `$HOME/db2_cli_odbc_driver1` and `$HOME/db2_cli_odbc_driver2`, where you will install the driver.
2. Locate the compressed file that contains the driver on the install CD. In this scenario, the file would be called `db2_driver_for_odbc_cli.tar.Z`.
3. Copy `db2_driver_for_odbc_cli.tar.Z` to the install directories, `$HOME/db2_cli_odbc_driver1` and `$HOME/db2_cli_odbc_driver2`.
4. Uncompress `db2_driver_for_odbc_cli.tar.Z` in each directory:

```
cd $HOME/db2_cli_odbc_driver1
uncompress db2_driver_for_odbc_cli.tar.Z
tar -xvf db2_driver_for_odbc_cli.tar
cd $HOME/db2_cli_odbc_driver2
uncompress db2_driver_for_odbc_cli.tar.Z
tar -xvf db2_driver_for_odbc_cli.tar
```
5. Delete `db2_driver_for_odbc_cli.tar.Z`.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Obtaining the IBM DB2 Driver for ODBC and CLI” on page 243
- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Intalling the IBM DB2 Driver for ODBC and CLI on a machine with an existing DB2 client” on page 247

Intalling the IBM DB2 Driver for ODBC and CLI on a machine with an existing DB2 client

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed separately.

You can install one or more copies of the IBM DB2 Driver for ODBC and CLI on a machine where a DB2 Client or DB2 Runtime Client is already installed. You might want to do this if you have developed some ODBC or CLI database applications with the DB2 Client that you plan to deploy with the IBM DB2 Driver for ODBC and CLI, because it enables you to test the database applications with the driver on the same machine as your development environment.

Prerequisites:

To install the IBM DB2 Driver for ODBC and CLI on the same machine as a DB2 Client or DB2 Runtime Client, you need:

- to obtain the compressed file that contains the driver.
See: Obtaining the IBM DB2 Driver for ODBC and CLI.

Procedure:

The procedure for installing one or more copies of the IBM DB2 Driver for ODBC and CLI on a machine that already has a DB2 Client or DB2 Runtime Client installed is the same as the procedure for installing the driver on a machine that has no DB2 Client installed.

See: Installing the IBM DB2 Driver for ODBC and CLI and Installing multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Obtaining the IBM DB2 Driver for ODBC and CLI” on page 243
- “Installing multiple copies of the IBM DB2 Driver for ODBC and CLI on the same machine” on page 246
- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245

Chapter 30. Configuring the IBM DB2 Driver for ODBC and CLI

Configuring the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

You must configure the IBM DB2 Driver for ODBC and CLI, and the software components of your database application runtime environment in order for your applications to use the driver successfully.

Prerequisites:

To configure the IBM DB2 Driver for ODBC and CLI and your application environment for the driver, you need:

- one or more copies of the driver installed.

See: Installing the IBM DB2 Driver for ODBC and CLI.

Procedure:

To configure the IBM DB2 Driver for ODBC and CLI, and the runtime environment of your DB2 CLI and ODBC applications to use the driver:

1. Configure aspects of the driver's behaviour such as data source name, user name, performance options, and connection options by updating the `db2cli.ini` initialization file.

See: `db2cli.ini` initialization file

There is no support for the Command Line Processor (CLP) with the IBM DB2 Driver for ODBC and CLI. For this reason, you can not update CLI configuration using the CLP command "db2 update CLI cfg"; you must update the `db2cli.ini` initialization file manually.

The `db2cli.ini` file is located in the install directory on Windows operating systems, and in the `cfg` subdirectory of the install directory on other operating systems.

If you have multiple copies of the IBM DB2 Driver for ODBC and CLI installed, each copy of the driver will have its own `db2cli.ini` file. Ensure you make the additions to the `db2cli.ini` for the correct copy of the driver.

2. Configure application environment variables.

See: Configuring environment variables for the IBM DB2 Driver for ODBC and CLI

3. For applications participating in transactions managed by the Microsoft Distributed Transaction Coordinator (DTC) only: you must register the driver with the DTC.

See: Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft DTC.

4. For ODBC applications using the Microsoft ODBC driver manager only: you must register the driver with the Microsoft driver manager.

See: Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager.

Related concepts:

- “db2cli.ini initialization file” on page 277
- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Configuring environment variables for the IBM DB2 Driver for ODBC and CLI” on page 250
- “Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft DTC” on page 253
- “Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager” on page 254

Configuring environment variables for the IBM DB2 Driver for ODBC and CLI

Configuring environment variables for the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

To use the IBM DB2 Driver for ODBC and CLI, there are two types of environment variables that you might have to set: environment variables that have replaced some DB2 registry variables; and an environment variable that tells your applications where to find the driver libraries.

Prerequisites:

To configure environment variables for the IBM DB2 Driver for ODBC and CLI, you need:

- one or more copies of the driver installed.
See: Installing the IBM DB2 Driver for ODBC and CLI.

Procedure:

To configure environment variables for the IBM DB2 Driver for ODBC and CLI:

1. Optional: set any applicable DB2 environment variables which have DB2 registry variables.

There is no support for the Command Line Processor (CLP) with the IBM DB2 Driver for ODBC and CLI. For this reason, you can not configure DB2 registry variables using the “db2set” CLP command. Required DB2 registry variables have been replaced with environment variables.

For more information about DB2 registry variables, see: DB2 registry and environment variables.

For a list of the environment variables that can be used instead of DB2 registry variables, see: Environment variables supported by the IBM DB2 Driver for ODBC and CLI.

2. Required for AIX 5.2 operating systems only: Set the local environment variable `DB2_CLI_DRIVER_INSTALL_PATH` to the directory in which the driver is

installed. (If there are multiple copies of the IBM DB2 Driver for ODBC and CLI installed, ensure DB2_CLI_DRIVER_INSTALL_PATH points to the intended copy of the driver.)

3. Optional: set the environment variable LIBPATH (on AIX systems) or LD_LIBRARY_PATH (on UNIX systems) to the directory in which the driver is installed. (If there are multiple copies of the IBM DB2 Driver for ODBC and CLI installed, ensure LIBPATH or LD_LIBRARY_PATH points to the intended copy of the driver.)

This step is not necessary if your applications statically link to, or dynamically load the driver's library (db2cli.dll on Windows systems, or libdb2.a on other systems) with the fully qualified name.

You are recommended to dynamically load the library using the fully qualified library name. On Windows operating systems, you are recommended to use the LoadLibraryEx method.

Restrictions:

If there are multiple versions of the IBM DB2 Driver for ODBC and CLI installed on the same machine, or if there are other DB2 Version 9 products installed on the same machine, setting environment variables (for example, setting LIBPATH or LD_LIBRARY_PATH to point to the IBM DB2 Driver for ODBC and CLI library) might break existing applications. When setting an environment variable, ensure that it is appropriate for all applications running in the scope of that environment.

Related concepts:

- “DB2 registry and environment variables” in *Performance Guide*

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245

Related reference:

- “Environment variables supported by the IBM DB2 Driver for ODBC and CLI” on page 251

Environment variables supported by the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

The IBM DB2 Driver for ODBC and CLI does not support the command line processor (CLP.) This means that the usual mechanism to set DB2 registry variables, using the “db2set” CLP command, is not possible. Relevant DB2 registry variables will be supported with the IBM DB2 Driver for ODBC and CLI as environment variables instead.

The DB2 registry variables that will be supported by the IBM DB2 Driver for ODBC and CLI as environment variables are:

Table 21. DB2 registry variables supported as environment variables

Type of variable	Variable name(s)
General variables See: General registry variables.	DB2ACCOUNT DB2BIDI DB2CODEPAGE DB2COUNTRY DB2GRAPHICUNICODESERVER DB2LOCALE DB2TERRITORY
System environment variables See: System environment variables.	DB2DOMAINLIST
Communications variables See: Communications variables.	DB2_FORCE-NLS_CACHE DB2SORCVBUF DB2SOSNDBUF DB2TCP_CLIENT_RCVTIMEOUT
Performance variables See: Performance variables.	DB2_NO_FORK_CHECK
Miscellaneous variables See: Miscellaneous variables.	DB2CLIINIPATH DB2_ENABLE_LDAP DB2LDAP_BASEDN DB2LDAP_CLIENT_PROVIDER DB2LDAPHOST DB2LDAP_KEEP_CONNECTION DB2LDAP_SEARCH_SCOPE DB2NOEXITLIST
Diagnostic variables See: DiagPath CLI/ODBC configuration keyword.	DB2_DIAGPATH

Related concepts:

- “DB2 registry and environment variables” in *Performance Guide*

Related tasks:

- “Configuring environment variables for the IBM DB2 Driver for ODBC and CLI” on page 250

db2oreg1.exe overview

The utility `db2oreg1.exe` is used on Windows operating systems only. This utility does two things: it registers the XA library of the IBM DB2 Driver for ODBC and CLI with the Microsoft Distributed Transaction Coordinator (DTC); and it registers the driver with the Microsoft ODBC driver manager.

Conditions requiring that you run the `db2oreg1.exe` utility:

You must run the `db2oreg1.exe` utility if:

- your applications that use the IBM DB2 Driver for ODBC and CLI will be participating in distributed transactions managed by the DTC; or
- your applications that use the IBM DB2 Driver for ODBC and CLI will be connecting to ODBC data sources.

When to run the db2oreg1.exe utility:

If you use the db2oreg1.exe utility, you must run it when:

- you install the IBM DB2 Driver for ODBC and CLI; and
- you uninstall the IBM DB2 Driver for ODBC and CLI.

The db2oreg1.exe utility makes changes to the Windows registry when you run it after installing the driver. If you uninstall the driver, you should run the utility again to undo those changes.

How to run the db2oreg1.exe utility:

- db2oreg1.exe is located in bin subdirectory where the IBM DB2 Driver for ODBC and CLI is installed.
- To list the parameters the db2oreg1.exe utility takes, and how to use them, run the utility with the “-h” option.

Related concepts:

- “Supported transaction managers” in *Getting Started with Database Application Development*

Related tasks:

- “Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft DTC” on page 253

Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft DTC

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

To use the IBM DB2 Driver for ODBC and CLI with database applications that participate in transactions managed by the Microsoft Distributed Transaction Coordinator (DTC), you must register the driver with the DTC.

Here is a link to the Microsoft article outlining the details of this security requirement:

[Registry Entries Are Required for XA Transaction Support](#)

Prerequisites:

To register the IBM DB2 Driver for ODBC and CLI with the DTC, you need:

- one or more copies of the driver installed.

See: [Installing the IBM DB2 Driver for ODBC and CLI](#).

Restrictions:

You only need to register the IBM DB2 Driver for ODBC and CLI with the DTC if your applications that use the driver are participating in transactions managed by the DTC.

Procedure:

To register the IBM DB2 Driver for ODBC and CLI with the DTC, do the following for each copy of the driver that is installed:

- run the db2oreg1.exe utility.

See: db2oreg1.exe overview.

The db2oreg1.exe utility makes changes to the Windows registry when you run it after installing the driver. If you uninstall the driver, you should run the utility again to undo those changes.

Related concepts:

- “db2oreg1.exe overview” on page 252
- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Configuring the IBM DB2 Driver for ODBC and CLI” on page 249

Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

For ODBC applications to use the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager, you must register the driver with the driver manager.

Prerequisites:

To register the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager, you need:

- one or more copies of the driver installed.

See: Installing the IBM DB2 Driver for ODBC and CLI.

Restrictions:

The Microsoft ODBC driver manager is the only ODBC driver manager with which you must register the IBM DB2 Driver for ODBC and CLI. The other ODBC driver managers do not require this.

Procedure:

To register the IBM DB2 Driver for ODBC and CLI with the Microsoft driver manager, do the following for each copy of the driver that is installed:

- run the db2iodbc.exe utility.

See: db2oreg1.exe overview.

The db2oreg1.exe utility makes changes to the Windows registry when you run it after installing the driver. If you uninstall the driver, you should run the utility again to undo those changes.

Related concepts:

- “db2oreg1.exe overview” on page 252

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Configuring the IBM DB2 Driver for ODBC and CLI” on page 249
- “Registering ODBC data sources for applications using the IBM DB2 Driver for ODBC and CLI” on page 259

Chapter 31. Connecting to databases with the IBM DB2 Driver for ODBC and CLI

Connecting to databases with the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

The IBM DB2 Driver for ODBC and CLI does not create a local database directory. This means that when you use this driver, you must make connectivity information available to your applications in other ways.

Prerequisites:

To connect to databases with the IBM DB2 Driver for ODBC and CLI, you need:

- database(s) to which to connect; and
- one or more copies of the driver installed.

See: Installing the IBM DB2 Driver for ODBC and CLI.

Procedure:

There are five ways to specify connectivity information so that your CLI and ODBC database applications can use the IBM DB2 Driver for ODBC and CLI to connect to a database.

To configure connectivity for a database when using the IBM DB2 Driver for ODBC and CLI, do one of:

1. Specify the database connectivity information in the connection string parameter to `SQLDriverConnect`.

See: `SQLDriverConnect` function (CLI) - (Expanded) Connect to a data source.

or

2. For CLI applications only: put the database connectivity information in the DB2 CLI configuration file.

See: `db2cli.ini` initialization file.

There is no support for the Command Line Processor (CLP) with the IBM DB2 Driver for ODBC and CLI. For this reason, you can not update CLI configuration using the CLP command `"db2 update CLI cfg"`; you must update the `db2cli.ini` initialization file manually.

The `db2cli.ini` file is located in the install directory on Windows operating systems, and in the `cfg` subdirectory of the install directory on other operating systems.

If you have multiple copies of the IBM DB2 Driver for ODBC and CLI installed, each copy of the driver will have its own `db2cli.ini` file. Ensure you make the additions to the `db2cli.ini` for the correct copy of the driver.

or

3. For ODBC applications only: register the database as an ODBC data source with the ODBC driver manager.

See: Registering ODBC data sources for applications using the IBM DB2 Driver for ODBC and CLI

or

4. Use the FileDSN CLI/ODBC keyword to identify a file DSN that contains the database connectivity information.

See: FileDSN CLI/ODBC configuration keyword.

A file DSN is a file that contains database connectivity information. You can create a file DSN by using the SaveFile CLI/ODBC keyword. On Microsoft Windows you can use the Microsoft ODBC driver manager to create a file DSN.

or

5. For local database servers only: use the PROTOCOL and DB2INSTANCE CLI/ODBC keywords to identify the local database:

- Set the PROTOCOL CLI/ODBC keyword to the value "Local"; and
- Set the DB2INSTANCE CLI/ODBC keyword to the instance name of the local database server on which the database is located.

See: Protocol CLI/ODBC configuration keyword.

Here is a list of CLI/ODBC keywords that work with file DSN or DSN-less connections:

- AltHostName CLI/ODBC configuration keyword;
- AltPort CLI/ODBC configuration keyword;
- Authentication CLI/ODBC configuration keyword;
- BIDI CLI/ODBC configuration keyword;
- FileDSN CLI/ODBC configuration keyword;
- Instance CLI/ODBC configuration keyword;
- Interrupt CLI/ODBC configuration keyword;
- KRBPlugin CLI/ODBC configuration keyword;
- Protocol CLI/ODBC configuration keyword;
- PWDPlugin CLI/ODBC configuration keyword;
- SaveFile CLI/ODBC configuration keyword;

Examples:

For the following examples, consider a database with the following properties:

- the database or subsystem is called db1 on the server;
- the server is located at 11.22.33.444;
- the access port is 56789; and
- the transfer protocol is TCP/IP.

To make a connection to the database in a CLI application, you could do one of the following:

1. Call SQLConnect with a connection string that contains:

```
“Database=db1;                               Protocol=tcip;
                                             Hostname=11.22.33.444;
                                             Servicename=56789;”
```

or

2. Add the following to `db2cli.ini`:

```
[db1]
Database=db1
Protocol=tcPIP
Hostname=11.22.33.444
Servicename=56789
```

To make a connection to the database in an ODBC application:

1. Register the database as an ODBC data source called `odbc_db1` with the driver manager; and
2. call `SQLConnect` with a connection string that contains:
“Database=odbc_db1;”

Related concepts:

- “db2cli.ini initialization file” on page 277
- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Registering ODBC data sources for applications using the IBM DB2 Driver for ODBC and CLI” on page 259
- “Using security plugins with the IBM DB2 Driver for ODBC and CLI” on page 261

Related reference:

- “FileDSN CLI/ODBC configuration keyword” on page 323
- “Protocol CLI/ODBC configuration keyword” on page 356
- “AltHostName CLI/ODBC configuration keyword” on page 284
- “AltPort CLI/ODBC configuration keyword” on page 285
- “Authentication CLI/ODBC configuration keyword” on page 290
- “BIDI CLI/ODBC configuration keyword” on page 291
- “Instance CLI/ODBC configuration keyword” on page 328
- “Interrupt CLI/ODBC configuration keyword” on page 329
- “KRBPlugin CLI/ODBC configuration keyword” on page 330
- “PWDPlugin CLI/ODBC configuration keyword” on page 358
- “SaveFile CLI/ODBC configuration keyword” on page 365
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*

Registering ODBC data sources for applications using the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

Before an ODBC database application can use the IBM DB2 Driver for ODBC and CLI to connect to a database, you must: 1) register the database as an ODBC data source with the ODBC driver manager; and 2) identify the IBM DB2 Driver for ODBC and CLI as the ODBC driver for the data source.

Prerequisites:

To register a database as an ODBC data source and associate the IBM DB2 Driver for ODBC and CLI with it, you need:

- databases to which your ODBC applications are going to connect;
- an ODBC driver manager installed; and
- one or more copies of the driver installed.

See: Installing the IBM DB2 Driver for ODBC and CLI.

Procedure:

The procedure for registering a database as an ODBC data source and associating the IBM DB2 Driver for ODBC and CLI with it depends on which driver manager you are using.

- For the Microsoft ODBC driver manager:
 1. Register the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager using the `db2oreg1.exe` utility.
See: Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager.
 2. Register the database as an ODBC data source.
See: Setting up the Windows CLI environment.
- For open source ODBC driver managers:
 1. Identify the database as an ODBC data source by adding database information to the `odbc.ini` file.
See: Setting up the UNIX ODBC environment.
 2. Associate the IBM DB2 Driver for ODBC and CLI with the data source by adding the fully qualified driver library file in the section of the `odbc.ini` file for the database.

The name of the IBM DB2 Driver for ODBC and CLI library file is `db2app.dll` on Windows platforms, and `db2app.lib` on other platforms. The driver library file is located in the "lib" subdirectory of the directory in which you installed the driver.

If you have multiple copies of the IBM DB2 Driver for ODBC and CLI installed, ensure the intended copy is identified in the `odbc.ini` file.

Example:

If you are registering ODBC data sources with an open source driver manager under the following conditions:

- the operating system on the target machine is a AIX;
- there are two copies of the IBM DB2 Driver for ODBC and CLI installed at
`$HOME/db2_cli_odbc_driver1` and
`$HOME/db2_cli_odbc_driver2`
- there are two ODBC database applications:
 1. `ODBCapp_A`
 - connects to two data sources, `db1` and `db2`; and
 - is intended to use the copy of the driver installed at
`$HOME/db2_cli_odbc_driver1`.
 2. `ODBCapp_B`
 - connects to the data source `db3`; and

- is intended to use the copy of the driver installed at
\$HOME/db2_cli_odbc_driver2.

add the following the entries in the odbc.ini file:

```
[db1]
Driver=$HOME/db2_cli_odbc_driver1/lib/libdb2.a
Description=First ODBC data source for ODBCapp1,
    using the first copy of the IBM DB2 Driver for ODBC and CLI

[db2]
Driver=$HOME/db2_cli_odbc_driver1/lib/libdb.a
Description=Second ODBC data source for ODBCapp1,
    using the first copy of the IBM DB2 Driver for ODBC and CLI

[db3]
Driver=$HOME/db2_cli_odbc_driver2/lib/libdb2.a
Description=First ODBC data source for ODBCapp2,
    using the second copy of the IBM DB2 Driver for ODBC and CLI
```

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Registering the IBM DB2 Driver for ODBC and CLI with the Microsoft ODBC driver manager” on page 254
- “Setting up the Windows CLI environment” on page 215
- “Setting up the UNIX ODBC environment” on page 210
- “Configuring the IBM DB2 Driver for ODBC and CLI” on page 249
- “Connecting to databases with the IBM DB2 Driver for ODBC and CLI” on page 257

Using security plugins with the IBM DB2 Driver for ODBC and CLI

A security plug-in is a dynamically-loadable library that provides authentication security services.

Procedure:

Using security plug-ins with the IBM DB2 Driver for ODBC and CLI is no different from using security plug-ins with a DB2 Client or DB2 Runtime Client.

For more information about using security plugins with DB2, see:

Security plug-ins.

When you read about using security plug-ins throughout the DB2 Information Center and DB2 books, consider the IBM DB2 Driver for ODBC and CLI like a DB2 client. Details about using security plug-ins with DB2 clients apply to using security plug-ins with the IBM DB2 Driver for ODBC and CLI too.

Related concepts:

- “Authentication” in *Administration Guide: Planning*
- “Security plug-ins” in *Administrative API Reference*

Chapter 32. Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI

Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either DB2 client.

The IBM DB2 Driver for ODBC and CLI provides runtime support for:

- the DB2 CLI application programming interface (API);
- the ODBC API;
- the XA API; and
- connecting to databases.

Prerequisites:

To run database applications with the IBM DB2 Driver for ODBC and CLI, you need:

- one or more copies of the driver installed.
See: Installing the IBM DB2 Driver for ODBC and CLI.
- to have configured the application environment for the driver.
See: Configuring the IBM DB2 Driver for ODBC and CLI.

Procedure:

When writing applications for, or migrating applications to using the IBM DB2 Driver for ODBC and CLI:

- Ensure your applications use only the DB2 CLI, ODBC and XA API functions that are supported by the driver.
See:
 - DB2 CLI and ODBC API support in the IBM DB2 Driver for ODBC and CLI
 - XA API support in the IBM DB2 Driver for ODBC and CLI
- Ensure your applications are not attempting to make use of DB2 Client or DB2 Runtime Client functionality that is restricted in the driver.
See: Restrictions of the IBM DB2 Driver for ODBC and CLI
- Use the 32-bit version of the driver with 32-bit database applications, and use the 64-bit version of the driver with 64-bit database applications.
- Understand the available tracing, logging, and diagnostic support provided by the driver for investigating problems.
See: Diagnostic support in the IBM DB2 Driver for ODBC and CLI

Related concepts:

- Chapter 27, "IBM DB2 Driver for ODBC and CLI overview," on page 241

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Configuring the IBM DB2 Driver for ODBC and CLI” on page 249

Related reference:

- “DB2 CLI and ODBC API support in the IBM DB2 Driver for ODBC and CLI” on page 264
- “Diagnostic support in the IBM DB2 Driver for ODBC and CLI” on page 266
- “LDAP support in the IBM DB2 Driver for ODBC and CLI” on page 265
- “Restrictions of the IBM DB2 Driver for ODBC and CLI” on page 266
- “XA API support in the IBM DB2 Driver for ODBC and CLI” on page 265

DB2 CLI and ODBC API support in the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either DB2 client.

The IBM DB2 Driver for ODBC and CLI supports the ANSI and (where they exist) the Unicode versions of the following ODBC and DB2 CLI functions:

SQLAllocConnect	SQLExtendedPrepare	SQLNumParams
SQLAllocEnv	SQLFetch	SQLNumResultCols
SQLAllocHandle	SQLFetchScroll	SQLParamData
SQLAllocStmt	SQLForeignKeys	SQLParamOptions
SQLBindCol	SQLFreeConnect	SQLPrepare
SQLBindFileToCol	SQLFreeEnv	SQLPrimaryKeys
SQLBindFileToParam	SQLFreeHandle	SQLProcedureColumns
SQLBindParameter	SQLFreeStmt	SQLProcedures
SQLBrowseConnect	SQLGetConnectAttr	SQLPutData
SQLBuildDataLink	SQLGetConnectOption	SQLRowCount
SQLBulkOperations	SQLGetCursorName	SQLSetColAttributes
SQLCancel	SQLGetData	SQLSetConnectAttr
SQLCloseCursor	SQLGetDataLinkAttr	SQLSetConnectOption
SQLColAttribute	SQLGetDescField	SQLSetConnection
SQLColAttributes	SQLGetDescRec	SQLSetCursorName
SQLColumnPrivileges	SQLGetDiagField	SQLSetDescField
SQLColumns	SQLGetDiagRec	SQLSetDescRec
SQLConnect	SQLGetEnvAttr	SQLSetEnvAttr
SQLCopyDesc	SQLGetFunctions	SQLSetParam
SQLDataSources	SQLGetInfo	SQLSetPos
SQLDescribeCol	SQLGetLength	SQLSetScrollOptions
SQLDescribeParam	SQLGetPosition	SQLSetStmtAttr
SQLDisconnect	SQLGetSQLCA	SQLSetStmtOption
SQLDriverConnect	SQLGetStmtAttr	SQLSpecialColumns
SQLEndTran	SQLGetStmtOption	SQLStatistics
SQLError	SQLGetSubString	SQLTablePrivileges
SQLExecDirect	SQLGetTypeInfo	SQLTables
SQLExecute	SQLMoreResults	SQLTransact
SQLExtendedBind	SQLNativeSql	
SQLExtendedFetch	SQLNextResult	

Related tasks:

- “Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI” on page 263

XA API support in the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either DB2 client.

The IBM DB2 Driver for ODBC and CLI supports the following XA API functions:

- xa_open
- xa_close
- xa_start
- xa_end
- xa_prepare
- xa_commit
- xa_rollback
- xa_forget
- xa_recover

Related tasks:

- “Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI” on page 263

LDAP support in the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either DB2 client.

The IBM DB2 Driver for ODBC and CLI supports the LDAP Database Directory support with one restriction:

- the LDAP cache can not be saved to disk; it is an in-memory cache only. The DB2LDAPCACHE registry variable is ignored.

The steps for configuring the database application environment to enable LDAP when using the IBM DB2 Driver for ODBC and CLI are the same as for other scenarios, except that the DB2LDAPCACHE registry variable is ignored.

For more information about enabling LDAP, see:

- Enabling LDAP support after installation is complete.

Related concepts:

- “Lightweight Directory Access Protocol (LDAP) overview” in *Administration Guide: Implementation*

Related tasks:

- “Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI” on page 263

Restrictions of the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either DB2 client.

The IBM DB2 Driver for ODBC and CLI provides runtime support for:

- the DB2 CLI application programming interface (API);
- the ODBC API;
- the XA API; and
- connecting to databases.

Functionality not supported by the IBM DB2 Driver for ODBC and CLI:

- CLI and ODBC application development;
- the Command Line Processor (CLP);
- administrative APIs;
- installation program.
 - You must install the driver manually.
See: Installing the IBM DB2 Driver for ODBC and CLI.
 - You must configure the driver manually.
See: Configuring the IBM DB2 Driver for ODBC and CLI.

Functionality supported with restrictions by the IBM DB2 Driver for ODBC and CLI:

- Messages will be reported only in English.
- There is no local database directory.
 - LDAP is supported, but the LDAP cache is not saved to disk.
See: LDAP support in the IBM DB2 Driver for ODBC and CLI.
 - Configuring for connectivity requires some additional steps.
See: Connecting to databases with the IBM DB2 Driver for ODBC and CLI.
- Not all diagnostic utilities are available.
See: Diagnostic support in the IBM DB2 Driver for ODBC and CLI.

Related tasks:

- “Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI” on page 263

Diagnostic support in the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either DB2 client.

The following tracing, logging, and diagnostic utilities are provided with the IBM DB2 Driver for ODBC and CLI:

CLI trace

The method for using CLI trace with the IBM DB2 Driver for ODBC and CLI is the same as the method for using CLI trace with a DB2 client.

DB2 trace

To turn on DB2 trace when you are using the IBM DB2 Driver for ODBC and CLI, you must invoke the `db2trc` utility from the “adm” subdirectory of the install directory of the driver.

For example, if you installed the driver at `$HOME/db2_cli_odbc_driver`, you must be in the `$HOME/db2_cli_odbc_driver/adm` directory when you invoke `db2trc`.

The IBM DB2 Driver for ODBC and CLI can be installed on a Network File System (NFS). If the driver is installed on a read-only NFS, the environment variable `DB2_DIAGPATH` must be set to allow DB2 trace to work.

db2diag.log

When you are using the IBM DB2 Driver for ODBC and CLI, the `db2diag.log` file will be located in the “db2dump” subdirectory of the driver install directory on UNIX and Linux operating systems, and in the “db2” subdirectory of the driver install directory on Windows operating systems.

db2support

The DB2 command line processor is not available with the IBM DB2 Driver for ODBC and CLI, so the CLP `db2support` utility is not available. However, an executable version of `db2support` will be available with the driver.

This executable version of `db2support` will collect the following information:

- `db2level` output;
- environment variables; and
- a listing of the contents of the IBM DB2 Driver for ODBC and CLI install directory.

You must invoke `db2support` from the “adm” subdirectory of the driver install directory.

For example, if you installed the driver at `$HOME/db2_cli_odbc_driver`, you must be in the `$HOME/db2_cli_odbc_driver/adm` directory when you invoke `db2support`.

Setting diagnostic options:

The IBM DB2 Driver for ODBC and CLI does not support the command line processor (CLP.) This means that the usual mechanism to set DB2 registry variables, using the `db2set` command, is not possible. However, the functionality of registry variables related to diagnostics is supported through CLI/ODBC keywords:

- `DiagLevel` CLI/ODBC configuration keyword
- `NotifyLevel` CLI/ODBC configuration keyword
- `DiagPath` CLI/ODBC configuration keyword

attributes for `SQLSetEnvAttr` and `SQLGetEnvAttr`:

- `SQL_ATTR_DIAGLEVEL`
- `SQL_ATTR_NOTIFY_LEVEL`
- `SQL_ATTR_DIAGPATH`

See: Environment attributes (CLI) list.

and environment variables:

- DB2_DIAGPATH

See: Environment variables supported by the IBM DB2 Driver for ODBC and CLI.

The CLI/ODBC keyword `DiagPath`, the attribute `SQL_ATTR_DIAGPATH`, and the environment variable `DB2_DIAGPATH` all have the same purpose: to specify where diagnostic output is located. There is one situation, however, where `DB2_DIAGPATH` must be used:

The IBM DB2 Driver for ODBC and CLI can be installed on a Network File System (NFS). If the driver is installed on a read-only NFS, the environment variable `DB2_DIAGPATH` must be set to allow DB2 trace to work.

Otherwise, the CLI/ODBC keyword `DiagPath`, the attribute `SQL_ATTR_DIAGPATH`, and the environment variable `DB2_DIAGPATH` all have the same effect.

Related concepts:

- “CLI trace files” in *Troubleshooting Guide*
- “Interpreting diagnostic log file entries” in *Troubleshooting Guide*
- “Obtaining a DB2 trace using `db2trc`” in *Troubleshooting Guide*

Related tasks:

- “Running DB2 CLI and ODBC applications using the IBM DB2 Driver for ODBC and CLI” on page 263

Chapter 33. Deploying the IBM DB2 Driver for ODBC and CLI with database applications

Deploying the IBM DB2 Driver for ODBC and CLI with database applications

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

You can simplify the deployment of your DB2 CLI and ODBC database applications by creating an install program. You can deploy the IBM DB2 Driver for ODBC and CLI with your DB2 CLI and ODBC database applications by incorporating the files and installation and configuration steps required for the driver into your install program.

Prerequisites:

To deploy the IBM DB2 Driver for ODBC and CLI with your applications you will need:

- a mechanism for deploying the applications, such as an install program;
- to obtain the compressed file that contains the driver;
See: Obtaining the IBM DB2 Driver for ODBC and CLI
- a redistribution license.

See: License requirements for the IBM DB2 Driver for ODBC and CLI.

Restrictions:

Under the terms of the redistribution licence, only some of the IBM DB2 Driver for ODBC and CLI files can be redistributed. Which files may be redistributed is listed in the file `redist.txt`. This file can be found in the compressed file that contains the driver, called `db2_driver_for_odbc_cli.zip` on the Windows platforms and `db2_driver_for_odbc_cli.tar.Z` on all other platforms.

Procedure:

To incorporate the IBM DB2 Driver for ODBC and CLI into your install program, you must do the following:

1. Copy the driver files into your install program. See the restrictions above about which driver files can be redistributed.
2. Set the install program to install the driver on the target machine.
See: Installing the IBM DB2 Driver for ODBC and CLI.
3. Set the install program to configure the environment on the target machine.
See: Configuring the IBM DB2 Driver for ODBC and CLI.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Obtaining the IBM DB2 Driver for ODBC and CLI” on page 243

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245
- “Configuring the IBM DB2 Driver for ODBC and CLI” on page 249

Related reference:

- “License requirements for the IBM DB2 Driver for ODBC and CLI” on page 270

License requirements for the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

You can download and install the IBM DB2 Driver for ODBC and CLI and use it with your ODBC and CLI applications without a special license. However, you need a redistribution license to redistribute the driver with your applications.

Under the terms of the redistribution licence, only some of the IBM DB2 Driver for ODBC and CLI files can be redistributed. Which files may be redistributed is listed in the file `redist.txt`. This file can be found in the compressed file that contains the driver, called `db2_driver_for_odbc_cli.zip` on the Windows platform and `db2_driver_for_odbc_cli.tar.Z` on all other platforms. For more information about where to find this file, see: [Obtaining the IBM DB2 Driver for ODBC and CLI](#)

The IBM DB2 ODBC and CLI Driver can be used to connect only to a properly licensed:

- DB2 for Linux, UNIX and Windows server;
- DB2 Connect server;
- WebSphere[®] Federation server;
- Cloudscape[™] Server;
- Informix[®] Database Server;
- DB2 for OS/390 and z/OS server (see below);
- DB2 for iSeries server (see below); and
- DB2 for VM/VSE server (see below).

The IBM DB2 ODBC and CLI Driver can be used to connect to DB2 for OS/390 and z/OS, DB2 for iSeries and DB2 for VM/VSE servers only if:

- a connection is established through a properly licensed DB2 Connect server; or
- directly to the server if and only if a properly formatted authentic license key file `db2con.lkf` is present. The file `db2con.lkf` is distributed as part of the DB2 Connect products, so the only way to obtain this license key file is to purchase one of the following DB2 Connect products:
 - DB2 Connect Personal Edition;
 - DB2 Connect Enterprise Edition;
 - DB2 Connect Application Server Edition;
 - DB2 Connect Unlimited Edition for zSeries;
 - DB2 Connect Unlimited Edition for iSeries;

No other product will provide this file or the license rights afforded by the presence of this file. Tampering with or unauthorized distribution of this file will constitute a breach of the license agreement.

Related concepts:

- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Chapter 34. Uninstalling the IBM DB2 Driver for ODBC and CLI

Uninstalling the IBM DB2 Driver for ODBC and CLI

The IBM DB2 Driver for ODBC and CLI is not part of the DB2 Client or the DB2 Runtime Client. It must be installed and configured separately.

Prerequisites:

To uninstall the IBM DB2 Driver for ODBC and CLI, you need:

- to ensure no applications are linked to the libraries of the copy of the driver that you will be uninstalling.

Procedure:

There is no uninstall utility for the IBM DB2 Driver for ODBC and CLI. It must be uninstalled manually:

- On Windows operating systems only:

If you ran the utility `db2iodbc.exe` to register the driver with the Microsoft Distributed Transaction Coordinator (DTC), you need to run the utility again before uninstalling the driver. This will clean up the Windows registry entries the utility created when you ran it to register the driver with the DTC.

see: `db2oreg1.exe` overview

- Remove the driver files by deleting the directory in which you installed the driver.

For example, if you installed the driver in `$HOME/db2_cli_odbc_driver`, then delete `$HOME/db2_cli_odbc_driver`.

Related concepts:

- “`db2oreg1.exe` overview” on page 252
- Chapter 27, “IBM DB2 Driver for ODBC and CLI overview,” on page 241

Related tasks:

- “Installing the IBM DB2 Driver for ODBC and CLI” on page 245

Part 5. CLI/ODBC configuration keywords

Chapter 35. CLI/ODBC configuration keywords

CLI/ODBC configuration keywords allow you to customize the behavior of the DB2 CLI driver. This chapter describes setting these keywords through the `db2cli.ini` initialization file and contains a listing of available configuration keywords.

db2cli.ini initialization file

The `db2cli.ini` initialization file contains various keywords and values that can be used to configure the behavior of DB2 CLI and the applications using it. The keywords are associated with the database *alias name*, and affect all DB2 CLI and ODBC applications that access the database.

By default, the location of the DB2 CLI/ODBC configuration keyword file is in the `sqllib` directory on Window platforms, and in the `sqllib/cfg` directory of the database instance running the CLI/ODBC applications on UNIX platforms. If the ODBC Driver Manager is used to configure a User Data Source on the Windows platform, a `db2cli.ini` might be created in the user's home (profile) directory.

The environment variable `DB2CLIINIPATH` can also be used to override the default and specify a different location for the file.

The configuration keywords enable you to:

- Configure general features such as data source name, user name, and password.
- Set options that will affect performance.
- Indicate query parameters such as wild card characters.
- Set patches or work-arounds for various ODBC applications.
- Set other, more specific features associated with the connection, such as code pages and IBM GRAPHIC data types.
- Override default connection options specified by an application. For example, if an application requests Unicode support from the CLI driver by setting the `SQL_ATTR_ANSI_APP` connection attribute, then setting `DisableUnicode=1` in the `db2cli.ini` file will force the CLI driver not to provide the application with Unicode support.

Note: If the CLI/ODBC configuration keywords set in the `db2cli.ini` file conflict with keywords in the `SQLDriverConnect()` connection string, then the `SQLDriverConnect()` keywords will take precedence.

The `db2cli.ini` initialization file is an ASCII file which stores values for the DB2 CLI configuration options. A sample file is shipped to help you get started. While most CLI/ODBC configuration keywords are set in the `db2cli.ini` initialization file, some keywords are set by providing the keyword information in the connection string to `SQLDriverConnect()` instead.

There is one section within the file for each database (data source) the user wishes to configure. If needed, there is also a common section that affects all database connections.

Only the keywords that apply to all database connections through the DB2 CLI/ODBC driver are included in the COMMON section. This includes the following keywords:

- CheckForFork
- DisableMultiThread
- JDBCTrace
- JDBCTraceFlush
- JDBCTracePathName
- QueryTimeoutInterval
- ReadCommonSectionOnNullConnect
- Trace
- TraceComm
- TraceErrImmediate
- TraceFileName
- TraceFlush
- TraceFlushOnError
- TraceLocks
- TracePathName
- TracePIDList
- TracePIDTID
- TraceRefreshInterval
- TraceStmtOnly
- TraceTime
- TraceTimeStamp

All other keywords are to be placed in the database specific section, described below.

Note: Configuration keywords are valid in the COMMON section, however, they will apply to all database connections.

The COMMON section of the `db2cli.ini` file begins with:

```
[COMMON]
```

Before setting a common keyword it is important to evaluate its impact on all DB2 CLI/ODBC connections from that client. A keyword such as TRACE, for instance, will generate information on all DB2 CLI/ODBC applications connecting to DB2 on that client, even if you are intending to troubleshoot only one of those applications.

Each database specific section always begins with the name of the data source name (DSN) between square brackets:

```
[data source name]
```

This is called the *section header*.

The parameters are set by specifying a keyword with its associated keyword value in the form:

KeywordName =*keywordValue*

- All the keywords and their associated values for each database must be located below the database section header.

- If the database-specific section does not contain a DBAlias keyword, the data source name is used as the database alias when the connection is established. The keyword settings in each section apply only to the applicable database alias.
- The keywords are not case sensitive; however, their values can be if the values are character based.
- If a database is not found in the .INI file, the default values for these keywords are in effect.
- Comment lines are introduced by having a semicolon in the first position of a new line.
- Blank lines are permitted.
- If duplicate entries for a keyword exist, the first entry is used (and no warning is given).

The following is a sample .INI file with 2 database alias sections:

```
; This is a comment line.
[MYDB22]
AutoCommit=0
TableType="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CurrentSQLID=SAIID
TableType="'TABLE'"
SchemaList="'USER1',CURRENT SQLID,'USER2'"
```

Although you can edit the `db2cli.ini` file manually on all platforms, it is recommended that you use the Configuration Assistant if it is available on your platform or the `UPDATE CLI CONFIGURATION` command. You must add a blank line after the last entry if you manually edit the `db2cli.ini` file.

Related reference:

- “UPDATE CLI CONFIGURATION command” in *Command Reference*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “DBAlias CLI/ODBC configuration keyword” on page 313
- “Trace CLI/ODBC configuration keyword” on page 375
- “Miscellaneous variables” in *Performance Guide*
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*

CLI/ODBC configuration keywords listing by category

The CLI/ODBC configuration keywords can be divided into the following categories:

- “Compatibility Configuration Keywords” on page 280
- “Data Source Configuration Keywords” on page 280
- “Data Type Configuration Keywords” on page 280
- “Enterprise Configuration Keywords” on page 281
- “Environment Configuration Keywords” on page 281
- “File DSN Configuration Keywords” on page 281
- “Optimization Configuration Keywords” on page 282
- “Service Configuration Keywords” on page 282
- “Static SQL Configuration Keywords” on page 283

- “Transaction Configuration Keywords” on page 283

While most CLI/ODBC configuration keywords are set in the `db2cli.ini` initialization file, some keywords are set by providing the keyword information in the connection string to `SQLDriverConnect()` instead. These keywords are identified as such below.

Compatibility Configuration Keywords:

The **Compatibility** set of options is used to define DB2 behavior. They can be set to ensure that other applications are compatible with DB2.

- “CheckForFork CLI/ODBC configuration keyword” on page 294
- “CursorTypes CLI/ODBC configuration keyword” on page 308
- “DeferredPrepare CLI/ODBC configuration keyword” on page 316
- “DescribeCall CLI/ODBC configuration keyword” on page 317
- “DescribeParam CLI/ODBC configuration keyword” on page 320
- “DisableKeysetCursor CLI/ODBC configuration keyword” on page 321
- “DisableMultiThread CLI/ODBC configuration keyword” on page 322
- “DisableUnicode CLI/ODBC configuration keyword” on page 322
- “OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword” on page 347
- “OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword” on page 348
- “RetCatalogAsCurrServer CLI/ODBC configuration keyword” on page 362
- “RetOleDbConnStr CLI/ODBC configuration keyword” on page 363
- “Trusted_Connection CLI/ODBC configuration keyword” on page 386 (use `SQLDriverConnect()` to set this keyword)

Data Source Configuration Keywords:

General keywords.

- “DBAlias CLI/ODBC configuration keyword” on page 313
- “PWD CLI/ODBC configuration keyword” on page 357
- “UID CLI/ODBC configuration keyword” on page 388

Data Type Configuration Keywords:

The **Data Type** set of options is used to define how DB2 reports and handles various data types.

- “BitData CLI/ODBC configuration keyword” on page 292
- “CurrentImplicitXMLParseOption CLI/ODBC configuration keyword” on page 302
- “DateTimeStringFormat CLI/ODBC configuration keyword” on page 310
- “DecimalFloatRoundingMode CLI/ODBC configuration keyword” on page 315
- “FloatPrecRadix CLI/ODBC configuration keyword” on page 324
- “Graphic CLI/ODBC configuration keyword” on page 326
- “LOBMaxColumnSize CLI/ODBC configuration keyword” on page 332
- “LongDataCompat CLI/ODBC configuration keyword” on page 333
- “MapBigintCDefault CLI/ODBC configuration keyword” on page 334

- “MapCharToWChar CLI/ODBC configuration keyword” on page 335
- “MapDateCDefault CLI/ODBC configuration keyword” on page 336
- “MapDateDescribe CLI/ODBC configuration keyword” on page 337
- “MapDecimalFloatDescribe CLI/ODBC configuration keyword” on page 338
- “MapGraphicDescribe CLI/ODBC configuration keyword” on page 339
- “MapTimeCDefault CLI/ODBC configuration keyword” on page 340
- “MapTimeDescribe CLI/ODBC configuration keyword” on page 340
- “MapTimestampCDefault CLI/ODBC configuration keyword” on page 342
- “MapTimestampDescribe CLI/ODBC configuration keyword” on page 342
- “MapXMLCDefault CLI/ODBC configuration keyword” on page 343
- “MapXMLDescribe CLI/ODBC configuration keyword” on page 344
- “OleDbReturnCharAsWChar CLI/ODBC configuration keyword” on page 348
- “XMLDeclaration CLI/ODBC configuration keyword” on page 392

Enterprise Configuration Keywords:

The **Enterprise** set of options is used to maximize the efficiency of connections to large databases.

- “ConnectNode CLI/ODBC configuration keyword” on page 299
- “CurrentPackagePath CLI/ODBC configuration keyword” on page 304
- “CurrentPackageSet CLI/ODBC configuration keyword” on page 305
- “CurrentRefreshAge CLI/ODBC configuration keyword” on page 306
- “CurrentSchema CLI/ODBC configuration keyword” on page 306
- “CurrentSQLID CLI/ODBC configuration keyword” on page 306
- “DBName CLI/ODBC configuration keyword” on page 314
- “GranteeList CLI/ODBC configuration keyword” on page 324
- “GrantorList CLI/ODBC configuration keyword” on page 325
- “ReportPublicPrivileges CLI/ODBC configuration keyword” on page 362
- “SchemaList CLI/ODBC configuration keyword” on page 365
- “ServerMsgMask CLI/ODBC configuration keyword” on page 366
- “SysSchema CLI/ODBC Configuration Keyword” on page 372
- “TableType CLI/ODBC configuration keyword” on page 374
- “UseServerMsgSP CLI/ODBC configuration keyword” on page 391

Environment Configuration Keywords:

The **Environment** set of options is used to define environment-specific settings, such as the location of various files on the server and client machines.

- “ConnectTimeout CLI/ODBC configuration keyword” on page 300
- “CurrentFunctionPath CLI/ODBC configuration keyword” on page 302
- “QueryTimeoutInterval CLI/ODBC configuration keyword” on page 358
- “ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword” on page 359
- “ReceiveTimeout CLI/ODBC configuration keyword” on page 360
- “TempDir CLI/ODBC configuration keyword” on page 374

File DSN Configuration Keywords:

The **File DSN** set of options is used to set the TCP/IP settings for a file DSN connection.

- “Database CLI/ODBC configuration keyword” on page 309
- “Hostname CLI/ODBC configuration keyword” on page 327
- “Port CLI/ODBC configuration keyword” on page 355
- “Protocol CLI/ODBC configuration keyword” on page 356
- “ServiceName CLI/ODBC configuration keyword” on page 366

Optimization Configuration Keywords:

The **Optimization** set of options is used to speed up and reduce the amount of network flow between the CLI/ODBC Driver and the server.

- “AllowGetDataLOBReaccess CLI/ODBC configuration keyword” on page 284
- “AppendForFetchOnly CLI/ODBC configuration keyword” on page 287
- “AppUsesLOBLocator CLI/ODBC configuration keyword” on page 287
- “BlockForNRows CLI/ODBC configuration keyword” on page 292
- “BlockLobs CLI/ODBC configuration keyword” on page 293
- “ClientBuffersUnboundLOBS CLI/ODBC configuration keyword” on page 296
- “CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword” on page 303
- “DB2Degree CLI/ODBC configuration keyword” on page 311
- “DB2Explain CLI/ODBC configuration keyword” on page 311
- “DB2NETNamedParam CLI/ODBC configuration keyword” on page 312
- “DB2Optimization CLI/ODBC configuration keyword” on page 313
- “DescribeInputOnPrepare CLI/ODBC configuration keyword” on page 318
- “DescribeOutputLevel CLI/ODBC configuration keyword” on page 318
- “KeepDynamic CLI/ODBC configuration keyword” on page 329
- “LOBCacheSize CLI/ODBC configuration keyword” on page 331
- “LOBFileThreshold CLI/ODBC configuration keyword” on page 332
- “LockTimeout CLI/ODBC configuration keyword” on page 333
- “MaxLOBBlockSize CLI/ODBC configuration keyword” on page 345
- “OptimizeForNRows CLI/ODBC configuration keyword” on page 349
- “Reopt CLI/ODBC configuration keyword” on page 360
- “SkipTrace CLI/ODBC configuration keyword” on page 367
- “StreamGetData CLI/ODBC configuration keyword” on page 371
- “StreamPutData CLI/ODBC configuration keyword” on page 372
- “Underscore CLI/ODBC configuration keyword” on page 389

Service Configuration Keywords:

The **Service** set of options is used to help in troubleshooting problems with CLI/ODBC connections. Some options can also be used by programmers to gain a better understanding of how their CLI programs are translated into calls to the server.

- “AppendAPIName CLI/ODBC configuration keyword” on page 285
- “AppendRowColToErrorMessage CLI/ODBC configuration keyword” on page 286
- “IgnoreWarnings CLI/ODBC configuration keyword” on page 327

- “IgnoreWarnList CLI/ODBC configuration keyword” on page 328
- “LoadXAInterceptor CLI/ODBC configuration keyword” on page 330
- “Patch1 CLI/ODBC configuration keyword” on page 350
- “Patch2 CLI/ODBC configuration keyword” on page 352
- “ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword” on page 361
- “RetryOnError CLI/ODBC configuration keyword” on page 364
- “ProgramName CLI/ODBC configuration keyword” on page 356
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceComm CLI/ODBC configuration keyword” on page 376
- “TraceErrImmediate CLI/ODBC configuration keyword” on page 377
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TraceFlushOnError CLI/ODBC configuration keyword” on page 379
- “TraceLocks CLI/ODBC configuration keyword” on page 380
- “TracePathName CLI/ODBC configuration keyword” on page 381
- “TracePIDList CLI/ODBC configuration keyword” on page 382
- “TracePIDTID CLI/ODBC configuration keyword” on page 383
- “TraceRefreshInterval CLI/ODBC configuration keyword” on page 383
- “TraceStmtOnly CLI/ODBC configuration keyword” on page 384
- “TraceTime CLI/ODBC configuration keyword” on page 385
- “TraceTimestamp CLI/ODBC configuration keyword” on page 386
- “WarningList CLI/ODBC configuration keyword” on page 391

Static SQL Configuration Keywords:

The **Static SQL** set of options is used when running static SQL statements in CLI/ODBC applications.

- “StaticCapFile CLI/ODBC configuration keyword” on page 368
- “StaticLogFile CLI/ODBC configuration keyword” on page 369
- “StaticMode CLI/ODBC configuration keyword” on page 370
- “StaticPackage CLI/ODBC configuration keyword” on page 370

Transaction Configuration Keywords:

The **Transaction** set of options is used to control and speed up SQL statements used in the application.

- “ArrayInputChain CLI/ODBC configuration keyword” on page 288
- “AsyncEnable CLI/ODBC configuration keyword” on page 289
- “AutoCommit CLI/ODBC configuration keyword” on page 290
- “ClientAcctStr CLI/ODBC configuration keyword” on page 294
- “ClientApplName CLI/ODBC configuration keyword” on page 295
- “ClientUserID CLI/ODBC configuration keyword” on page 297
- “ClientWrkStnName CLI/ODBC configuration keyword” on page 298
- “ConnectType CLI/ODBC configuration keyword” on page 301
- “CursorHold CLI/ODBC configuration keyword” on page 307
- “Mode CLI/ODBC configuration keyword” on page 346
- “SQLOverrideFileName CLI/ODBC configuration keyword” on page 368

- “TxnIsolation CLI/ODBC configuration keyword” on page 388
- “UseOldStpCall CLI/ODBC configuration keyword” on page 390

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Introduction to DB2 CLI and ODBC” on page 3

Related reference:

- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*

AllowGetDataLOBReaccess CLI/ODBC configuration keyword

Keyword description:

Specify whether the application can call `SQLGetData()` for previously accessed LOB columns when querying database servers that support Dynamic Data Format.

db2cli.ini keyword syntax:

`AllowGetDataLOBReaccess = 0 | 1`

Default setting:

Do not allow calls to `SQLGetData()` for previously accessed LOB columns when querying database servers that support Dynamic Data Format.

Usage notes:

This keyword only affects connections to database servers that support Dynamic Data Format. The default setting of 0 does not allow applications to call `SQLGetData()` for previously accessed LOB columns. Specify 1 to allow applications to call `SQLGetData()` for previously accessed LOB columns.

Note that when the keyword is set to 1 to allow re-access to LOB columns, some resources on the server might not be freed upon completion of `SQLGetData()`.

If the server does not support Dynamic Data Format, this keyword has no effect and calls to `SQLGetData()` for previously accessed LOB columns are allowed.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*

AltHostName CLI/ODBC configuration keyword

Keyword description:

Alternate host name to be used if the primary server specified by `HOSTNAME` cannot be contacted (Client Reroute.)

db2cli.ini keyword syntax:

`AltHostName = fully qualified alternate host name | IP address of node`

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies a fully qualified host name or the IP address of the node where the alternate server for the database resides.

If the primary server returns alternate server information, it will override this AltHostName setting. However, this keyword is read only. That means the db2cli.ini will not be updated with the alternate server information received from the primary server.

AltPort CLI/ODBC configuration keyword

Keyword description:

Alternate port to be used if the primary server specified by HOSTNAME and PORT cannot be contacted (Client Reroute.)

db2cli.ini keyword syntax:

AltPort = *port number*

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies the port number of the alternate server of the database manager instance where the alternate server for the database resides.

If the primary server returns alternate server information, it will override this AltPort setting. However, this keyword is read only. That means the db2cli.ini will not be updated with the alternate server information received from the primary server.

AppendAPIName CLI/ODBC configuration keyword

Keyword description:

Append the CLI/ODBC function name which generated an error to the error message text.

db2cli.ini keyword syntax:

AppendAPIName = 0 | 1

Default setting:

Do NOT display DB2 CLI function name.

Usage notes:

The DB2 CLI function (API) name that generated an error is appended to the error message retrieved using SQLGetDiagRec() or SQLError(). The function name is enclosed in curly braces { }.

For example,

```
[IBM][CLI Driver]" CLIxxxx: < text >  
SQLSTATE=XXXXX {SQLGetData}"  
0 = do NOT append DB2 CLI function name (default)  
1 = append the DB2 CLI function name
```

This keyword is only useful for debugging.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLError function (CLI) - Retrieve error information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record” in *Call Level Interface Guide and Reference, Volume 2*

AppendRowColToErrorMessage CLI/ODBC configuration keyword

Keyword description:

Append row and column numbers that generated the error to the error message string.

db2cli.ini keyword syntax:

AppendRowColToErrorMessage= 0 | 1

Default setting:

The default setting of 0 will return the error message string without the row and column numbers.

Usage notes:

Specify 1 to append the row and column number that generated the error to the error message string. The values for row and column numbers are only appended when DB2 CLI is able to apply a row or column number to the problem.

The row or column numbers appended to error messages are the same positive values that would be returned if an application called SQLGetDiagField() with the DiagIdentifier argument as SQL_DIAG_ROW_NUMBER or SQL_DIAG_COLUMN_NUMBER. When AppendRowColToErrorMessage is set to 1, errors returned from calls to SQLGetDescField(), SQLGetDescRec() or SQLError() will have these row or column numbers appended with the following format: Row=<r>, Col=<c>, if they can be determined.

For example, the default text for error CLI0111E is as follows:

```
[IBM][CLI Driver] CLI0111E Numeric value out of range. SQLSTATE=22003
```

Specifying 1 to append the row and column number will return the following text for error CLI0111E:

```
[IBM][CLI Driver] CLI0111E Numeric value out of range.  
SQLSTATE=22003 {Row=2,Col=1}
```

Note: It is also possible for an error to be returned with only a row number.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Header and record fields for the DiagIdentifier argument (CLI)” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLError function (CLI) - Retrieve error information” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDiagField function (CLI) - Get a field of diagnostic data” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

AppendForFetchOnly CLI/ODBC configuration keyword

Keyword description:

Control whether the clause FOR FETCH ONLY is appended to READ-ONLY SQL statements.

db2cli.ini keyword syntax:

AppendForFetchOnly = 0 | 1

Default setting:

The keyword is not set by default. DB2 CLI appends the "FOR FETCH ONLY" clause only when connected to certain server types.

Equivalent connection attribute:

SQL_ATTR_APPEND_FOR_FETCH_ONLY

Usage notes:

By default, DB2 CLI appends the "FOR FETCH ONLY" clause to read SELECT statements when connected to DB2 for z/OS or DB2 UDB for iSeries databases.

This keyword allows an application to control when DB2 CLI appends the "FOR FETCH ONLY" clause, for example, in a situation where an application is binding the DB2 CLI packages using different bind BLOCKING options (for example, BLOCKING UNAMBIG) and wants to suppress the blocking in order to keep positioned on a given row.

To change the default DB2 CLI behavior, the keyword can be set as follows:

- 0: DB2 CLI never appends the "FOR FETCH ONLY" clause to read SELECT statements regardless of the server type it is connecting to.
- 1: DB2 CLI always appends the "FOR FETCH ONLY" clause to read SELECT statements regardless of the server type it is connecting to.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

AppUsesLOBLocator CLI/ODBC configuration keyword

Keyword description:

Specify that applications are using LOB locators.

db2cli.ini keyword syntax:

AppUsesLOBLocator = 0 | 1

Default setting:

Applications are using LOB locators.

Equivalent connection or statement attribute:

SQL_ATTR_APP_USES_LOB_LOCATOR

Usage notes:

The default setting of 1 indicates that applications are using LOB locators. For applications that do not use LOB locators and are querying data on a server that supports Dynamic Data Format, specify 0 to indicate that LOB locators are not used and allow the return of LOB data to be optimized.

This keyword is ignored for stored procedure result sets.

If the keyword is set to 0 and an application binds a LOB locator to a result set using `SQLBindCol()`, an Invalid conversion error will be returned by the `SQLFetch()` function.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “LOB locators in CLI applications” on page 98

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

ArrayInputChain CLI/ODBC configuration keyword

Keyword description:

Enable array input without needing pre-specified size and memory allocation requirements of normal array input.

db2cli.ini keyword syntax:

ArrayInputChain = -1 | 0 | <positive integer>

Default setting:

Normal input array is enabled, where the array and its size must be specified before the corresponding `SQLExecute()` call is made.

Usage notes:

By default, array input (where an array of values is bound to an input parameter) requires the array and its size to be specified before the corresponding `SQLExecute()` function is called. An application, however, may not know the array size in advance, or the array size may be too large for the application to allocate from its pool of available memory. Under these circumstances, the application can set `ArrayInputChain=-1` and use the `SQL_ATTR_CHAINING_BEGIN` and `SQL_ATTR_CHAINING_END` statement attributes to enable chaining, which allows array input without the pre-specified size and memory requirements of normal array input.

To enable chaining:

1. Set the keyword `ArrayInputChain = -1`.
2. Prepare and bind input parameters to the SQL statement.
3. Set the `SQL_ATTR_CHAINING_BEGIN` statement attribute with `SQLSetStmtAttr()`.
4. Update the bound parameters with input data and call `SQLExecute()`.
5. Repeat Step 4 for as many rows as there are in the input array.
6. Set the `SQL_ATTR_CHAINING_END` statement attribute with `SQLSetStmtAttr()` after the last row in the array has been processed according to Step 4.

The effect of completing these steps will be the same as if normal array input had been used.

Setting `ArrayInputChain=0` (the default value) turns this array input feature off. `ArrayInputChain` can also be set to any positive integer which sets the array size to use for the input array.

Related concepts:

- “Reduction of network flows with CLI array input chaining” on page 62
- “db2cli.ini initialization file” on page 277

Related reference:

- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

AsyncEnable CLI/ODBC configuration keyword

Keyword description:

Enable or disable the ability to execute queries asynchronously.

db2cli.ini keyword syntax:

`AsyncEnable = 0 | 1`

Default setting:

Queries can be executed asynchronously.

Usage notes:

This option allows you to enable or disable support that allows queries to execute asynchronously. This only benefits applications that were written to take advantage of this feature by setting the `SQL_ATTR_ASYNC_ENABLE` attribute using `SQLSetStmtAttr()` or `SQLSetConnectAttr()`.

- 0 = Queries are not executed asynchronously
- 1 = Allow queries to be executed asynchronously. The application must also enable the asynchronous functionality by setting `SQL_ATTR_ASYNC_ENABLE` using `SQLSetStmtAttr()` or `SQLSetConnectAttr()`. (default)

Once a function has been called asynchronously, only the original function, `SQLAllocHandle()`, `SQLCancel()`, `SQLSetStmtAttr()`, `SQLGetDiagField()`, `SQLGetDiagRec()`, or `SQLGetFunctions()` can be called on the statement handle, until the original function returns a code other than

SQL_STILL_EXECUTING. Any other function called on any other statement handle under the same connection returns SQL_ERROR with an SQLSTATE of HY010 (Function sequence error).

Related concepts:

- “Asynchronous execution of CLI functions” on page 127
- “db2cli.ini initialization file” on page 277

Related tasks:

- “Executing functions asynchronously in CLI applications” on page 129

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Authentication CLI/ODBC configuration keyword

Keyword description:

A string value indicating the type of authentication to be used with file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

Authentication = **SERVER** | **SERVER_ENCRYPT** | **DATA_ENCRYPT** | **KERBEROS** | **GSSPLUGIN**

Default setting:

SERVER

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- Database;
- Protocol.

If Protocol=IPC, you need to set the following too:

- Instance.

If Protocol=TCPIP, you need to set the following too:

- Port; and
- Hostname.

If Kerberos is specified, then the KRBPlugin may also be optionally specified. If KRBPlugin is not specified, the default plugin IBMkrb5 will be used.

AutoCommit CLI/ODBC configuration keyword

Keyword description:

Specify whether the application commits each statement by default.

db2cli.ini keyword syntax:

AutoCommit = 1 | 0

Default setting:

Each statement is treated as a single, complete transaction.

Equivalent connection attribute:

SQL_ATTR_AUTOCOMMIT

Usage notes:

To be consistent with ODBC, DB2 CLI defaults with AutoCommit on, which means each statement is treated as a single, complete transaction. This keyword can provide an alternative default, but will only be used if the application does not specify a value for SQL_ATTR_AUTOCOMMIT.

1 = SQL_ATTR_AUTOCOMMIT_ON (default)

0 = SQL_ATTR_AUTOCOMMIT_OFF

Note: Most ODBC applications assume the default of AutoCommit to be on.

Extreme care must be used when overriding this default during runtime as the application may depend on this default to operate properly.

This keyword also allows you to specify whether autocommit should be enabled in a Distributed Unit of Work (DUOW) environment. If a connection is part of a coordinated Distributed Unit of Work, and AutoCommit is not set, the default does not apply; implicit commits arising from autocommit processing are suppressed. If AutoCommit is set to 1, and the connection is part of a coordinated Distributed Unit of Work, the implicit commits are processed. This may result in severe performance degradation, and possibly other unexpected results elsewhere in the DUOW system. However, some applications may not work at all unless this is enabled.

A thorough understanding of the transaction processing of an application is necessary, especially applications written by a third party, before applying it to a DUOW environment.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Multisite updates (two phase commit) in CLI applications” on page 137

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

BIDI CLI/ODBC configuration keyword

Keyword description:

Indicates the BIDI codepage when we are connected to a DB2 for z/OS.

db2cli.ini keyword syntax:

BIDI = *codepage*

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- Database;

- Protocol=TCPIP;
- Hostname; and
- Port.

BitData CLI/ODBC configuration keyword

Keyword description:

Specify whether binary data types are reported as binary or character data types.

db2cli.ini keyword syntax:

BitData = 1 | 0

Default setting:

Report FOR BIT DATA and BLOB data types as binary data types.

Usage notes:

This option allows you to specify whether ODBC binary data types (SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY, and SQL_BLOB), are reported as binary type data. IBM DBMSs support columns with binary data types by defining CHAR, VARCHAR, and LONG VARCHAR columns with the FOR BIT DATA attribute. DB2 Database for Linux, UNIX, and Windows will also support binary data via the BLOB data type (in this case it is mapped to a CLOB data type).

Only set BitData = 0 if you are sure that all columns defined as FOR BIT DATA or BLOB contain only character data, and the application is incapable of displaying binary data columns.

- 1 = report FOR BIT DATA and BLOB data types as binary data types (default).
- 0 = report FOR BIT DATA and BLOB data types as character data types.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQL symbolic and default data types for CLI applications” on page 41

BlockForNRows CLI/ODBC configuration keyword

Keyword description:

Specify the number of rows of data to be returned in a single fetch.

db2cli.ini keyword syntax:

BlockForNRows = <positive integer>

Default setting:

The server returns as many rows as can fit in a query block in a single fetch request.

Usage notes:

The BlockForNRows keyword controls the number of rows of data that are returned to the client in a single fetch request. If BlockForNRows is not specified (the default setting), then as many rows of non-LOB data as can fit in a query

block are returned from the server. If the result set contains LOB data, then the behavior `BlockForNRows` yields can be affected by the `BlockLobs` CLI/ODBC configuration keyword and the server's support for LOB blocking.

If `BlockForNRows` is not specified, `BlockLobs` is set to 1, and the server supports LOB blocking, then all LOB data associated with rows that fit completely within a single query block are returned in a single fetch request. LOB data is described here as being associated with a row, because the LOB data of a result set is itself not contained in the row. Instead, the row contains a reference to the actual LOB data.

If `BlockForNRows` is set to a positive integer `n`, then `n` rows of data will be returned in a single fetch request. If the result set contains LOB data and the server supports LOB blocking, then the LOB data that corresponds to the `n` rows of data will also be returned in the single fetch request. If the result set contains LOB data, but the server does not support LOB blocking, then only one row of data, including the LOB data, will be returned in a single fetch request.

Related concepts:

- “`db2cli.ini` initialization file” on page 277
- “Large object usage” in *SQL Guide*
- “Large object usage in CLI applications” on page 96

Related reference:

- “`BlockLobs` CLI/ODBC configuration keyword” on page 293
- “CLI/ODBC configuration keywords listing by category” on page 279

BlockLobs CLI/ODBC configuration keyword

Keyword description:

Enable LOB blocking fetch against servers that support LOB blocking.

db2cli.ini keyword syntax:

`BlockLobs = 0 | 1`

Default setting:

LOB blocking fetch is disabled.

Equivalent statement attribute:

`SQL_ATTR_BLOCK_LOBS`

Usage notes:

Setting `BlockLobs` to 1 enables all of the LOB data associated with rows that fit completely within a single query block to be returned in a single fetch request, if the server supports LOB blocking. LOB data is described here as being associated with a row, because the LOB data of a result set is itself not contained in the row. Instead, the row contains a reference to the actual LOB data. Therefore, with LOB blocking fetch, any rows of the result set that fit completely within the query block (where each row consists of non-LOB data, since LOB data is not stored directly in the row), will have their associated LOB data returned from the server, if the server supports LOB blocking.

If the server does not support LOB blocking and the result set contains LOB data, then only one row of data, including the LOB, will be returned in a single fetch request, irrespective of the value `BlockLobs` is set to.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Large object usage in CLI applications” on page 96
- “Large object usage” in *SQL Guide*

Related reference:

- “BlockForNRows CLI/ODBC configuration keyword” on page 292
- “CLI/ODBC configuration keywords listing by category” on page 279
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

CheckForFork CLI/ODBC configuration keyword

Keyword description:

Check for a forked process for each function call.

db2cli.ini keyword syntax:

0 | 1

Default setting:

DB2 CLI does not check for forked processes.

Usage notes:

DB2 CLI assumes that the process will never be forked. The CheckForFork keyword must be set to 1 if applications want to fork while connection and statement handles are allocated in order to avoid interfering with the parent process' active connections.

The SQL_ATTR_PROCESSCTL environment attribute can be set to SQL_PROCESSCTL_NOTHREAD option by an application to override the CheckForFork keyword for that application.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

ClientAcctStr CLI/ODBC configuration keyword

Keyword description:

Set client accounting string sent to host database.

db2cli.ini keyword syntax:

ClientAcctStr = *accounting string*

Default setting:

None

Only applicable when:

Connected to a host database using DB2 Connect

Equivalent environment or connection attribute:

SQL_ATTR_INFO_ACCTSTR

Usage notes:

This option allows the CLI application to set the client accounting string that is sent to the host database through DB2 Connect. Applications that do not offer the accounting string by default can take advantage of this keyword to provide this information.

Note the following conditions:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 200 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “ClientAppName CLI/ODBC configuration keyword” on page 295
- “ClientUserID CLI/ODBC configuration keyword” on page 297
- “ClientWrkStnName CLI/ODBC configuration keyword” on page 298

ClientAppName CLI/ODBC configuration keyword

Keyword description:

Set client application name sent to host database.

db2cli.ini keyword syntax:

ClientAppName = *application name*

Default setting:

None

Only applicable when:

Connected to a host database using DB2 Connect

Equivalent environment or connection attribute:

SQL_ATTR_INFO_APPLNAME

Usage notes:

This option allows the CLI application to set the client application name that is sent to the host database through DB2 Connect. Applications that do not offer the application name by default can take advantage of this keyword to provide this information.

Note the following conditions:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 32 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “ClientAcctStr CLI/ODBC configuration keyword” on page 294
- “ClientUserID CLI/ODBC configuration keyword” on page 297
- “ClientWrkStnName CLI/ODBC configuration keyword” on page 298

ClientBuffersUnboundLOBS CLI/ODBC configuration keyword

Keyword description:

Fetch LOB data instead of the LOB locator for LOB columns that have not been bound to application parameters.

db2cli.ini keyword syntax:

ClientBuffersUnboundLOBS = 0 | 1

Default setting:

A LOB locator is retrieved instead of the actual LOB data for LOB columns that have not been bound to application parameters.

Usage notes:

By default, when a result set contains a LOB column that has not been bound to an application parameter, DB2 CLI will fetch the corresponding LOB locator rather than the LOB data itself. The application must then use the `SQLGetLength()`, `SQLGetPosition()`, and `SQLGetSubString()` CLI functions to retrieve the LOB data. If the application regularly wants to retrieve the LOB data, then this default two-step process is unnecessary and could decrease performance. In this case, set `ClientBuffersUnboundLOBS = 1` to force DB2 CLI to fetch the LOB data instead of the LOB locator.

Servers that support Dynamic Data Format optimize the return of LOB and XML data depending on the actual length of the data. The LOB and XML data can be returned in its entirety, or as an internal token called a progressive reference. DB2 CLI manages progressive reference data retrieval.

For applications that are querying data on a server that supports Dynamic Data Format, setting the `LOBCacheSize` keyword sets a threshold that is used to determine if the data is returned in its entirety, or as a progressive reference. If the data has a length greater than the `LOBCacheSize` threshold value, the progressive reference will be returned to DB2 CLI to manage,

but if the data has a length less than or equal to the LOBCacheSize threshold value, the data will be returned in its entirety. Setting ClientBuffersUnboundLOBS to 1 is equivalent to setting LOBCacheSize to 2147483647 and will force the server to return the data in its entirety rather than as a progressive reference.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “LOB locators in CLI applications” on page 98

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “LOBCacheSize CLI/ODBC configuration keyword” on page 331
- “SQLGetLength function (CLI) - Retrieve length of a string value” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetPosition function (CLI) - Return starting position of string” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetSubString function (CLI) - Retrieve portion of a string value” in *Call Level Interface Guide and Reference, Volume 2*

ClientUserID CLI/ODBC configuration keyword

Keyword description:

Set client user ID sent to host database.

db2cli.ini keyword syntax:

ClientUserID = *userid*

Default setting:

None

Only applicable when:

Connected to a host database using DB2 Connect

Equivalent environment or connection attribute:

SQL_ATTR_INFO_USERID

Usage notes:

This option allows the CLI application to set the client user ID (accounting user ID) that is sent to the host database through DB2 Connect. Applications that do not offer the user ID by default can take advantage of this keyword to provide this information.

Note the following conditions:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 16 characters.
- This user ID is not to be confused with the authentication user ID. This user ID is for identification purposes only and is not used for any authorization.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “ClientAcctStr CLI/ODBC configuration keyword” on page 294
- “ClientApplName CLI/ODBC configuration keyword” on page 295
- “ClientWrkStnName CLI/ODBC configuration keyword” on page 298

ClientWrkStnName CLI/ODBC configuration keyword

Keyword description:

Set client workstation name sent to host database.

db2cli.ini keyword syntax:

ClientWrkStnName = *workstation name*

Default setting:

None

Only applicable when:

Connected to a host database using DB2 Connect

Equivalent environment or connection attribute:

SQL_ATTR_INFO_WRKSTNNAME

Usage notes:

This option allows the CLI application to set the client workstation name that is sent to the host database through DB2 Connect. Applications that do not offer the client workstation name by default can take advantage of this keyword to provide this information.

Note the following conditions:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 18 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “ClientAcctStr CLI/ODBC configuration keyword” on page 294
- “ClientApplName CLI/ODBC configuration keyword” on page 295

- “ClientUserID CLI/ODBC configuration keyword” on page 297

CLIPkg CLI/ODBC configuration keyword

Keyword description:

Specifies the number of large packages to be generated.

db2cli.ini keyword syntax:

CLIPkg = 3 | 4 | ... | 30

Default setting:

Three large packages are generated.

Usage notes:

This keyword is used to increase the number of sections for SQL statements in CLI/ODBC applications. If it is used, the administrator should explicitly bind the required bind files with the CLIPkg bind option. For client applications, the db2cli.ini file on the client must be updated with this value of CLIPkg. For CLI/JDBC stored procedures, the db2cli.ini file on the server (DB2 UDB Version 6.1 or later on UNIX or Intel™ platforms) must be updated with the same value of CLIPkg.

If the value is NOT an integer between 3 and 30, the default will be used without error or warning.

This setting only applies to large packages (containing 384 sections). The number of small packages (containing 64 sections) is 3 and cannot be changed.

It is recommended that you only increase the number of sections enough to run your application as the packages take up space in the database.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Handles in CLI” on page 15

Related reference:

- “BIND command” in *Command Reference*
- “CLI/ODBC configuration keywords listing by category” on page 279

ConnectNode CLI/ODBC configuration keyword

Keyword description:

Specify the database partition server to which a connection is to be made.

db2cli.ini keyword syntax:

ConnectNode = **integer value from 0 to 999** |
SQL_CONN_CATALOG_NODE

Default setting:

Database partition server which is defined with port 0 on the machine is used.

Only applicable when:

Connecting to a partitioned database environment.

Equivalent connection attribute:
SQL_ATTR_CONNECT_NODE

Usage notes:

Used to specify the target database partition server that you want to connect to. This keyword (or attribute setting) overrides the value of the environment variable DB2NODE. Can be set to:

- an integer between 0 and 999
- SQL_CONN_CATALOG_NODE

If this variable is not set, the target defaults to the database partition server that is defined with port 0 on the machine.

Note: This keyword does not affect the Control Center. The Control Center always connects to the catalog partition referred to by the SQL_CONN_CATALOG_NODE setting.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related tasks:

- “Setting environment variables on Windows” in *Administration Guide: Implementation*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

ConnectTimeout CLI/ODBC configuration keyword

Keyword description:

Specify the time in seconds to wait for a reply when trying to establish a connection to a server before terminating the attempt and generating a communication timeout.

db2cli.ini keyword syntax:

ConnectTimeout = 0 | 1 | 2 | ... | 32767

Default setting:

The client waits indefinitely for a reply from the server when trying to establish a connection.

Equivalent connection attribute:

SQL_ATTR_LOGIN_TIMEOUT

Usage notes:

If ConnectTimeout is set and client reroute is enabled, a connection will be attempted only once to the original server and once to the alternate server. Since the ConnectTimeout value is used when attempting to connect to each server, the maximum waiting time will be approximately double the specified value for ConnectTimeout. If neither server can be reached within the amount of time specified by the keyword, the following error message will be received:

```
SQL30081N A communication error has been detected. Communication
protocol being used: "TCP/IP". Communication API being used:
"SOCKETS". Location where the error was detected: "<ip address>".
Communication function detecting the error: "<failing function>".
Protocol specific error code(s): "<error code>", "*", "*".
SQLSTATE=08001
```

If ConnectTimeout is set and Sysplex exploitation is enabled, a connection will be attempted only once for each of the Sysplex members. Since the ConnectTimeout value is used when attempting to connect to each Sysplex member, the maximum waiting time will be approximately equal to the number of Sysplex members, times the amount of time specified by the ConnectTimeout keyword.

ConnectTimeout only applies to the TCPIP protocol and is not supported for connections to databases cataloged on a SOCKS-enabled TCP/IP node.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Automatic client reroute and high availability disaster recovery (HADR)” in *Data Recovery and High Availability Guide and Reference*
- “DB2 Sysplex exploitation” in *DB2 Connect User’s Guide*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CATALOG TCPIP/TCPIP4/TCPIP6 NODE command” in *Command Reference*

ConnectType CLI/ODBC configuration keyword

Keyword description:

Controls whether the application is to operate in a remote or distributed unit of work.

db2cli.ini keyword syntax:

ConnectType = 1 | 2

Default setting:

Remote unit of work.

Equivalent environment or connection attribute:

SQL_ATTR_CONNECTTYPE

Usage notes:

This option allows you to specify the default connect type. The options are:

- 1 = Remote unit of work. Multiple concurrent connections, each with its own commit scope. The concurrent transactions are not coordinated. This is the default.
- 2= Distributed unit of work. Coordinated connections where multiple databases participate under the same distributed unit of work.

The first connection determines the connect type for all other connections that are allocated under the same environment handle.

This keyword takes precedence over the environment or connection attribute.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

CurrentFunctionPath CLI/ODBC configuration keyword

Keyword description:

Specify the schema used to resolve function references and data type references in dynamic SQL statements.

db2cli.ini keyword syntax:

CurrentFunctionPath = *current_function_path*

Default setting:

See description below.

Usage notes:

This keyword defines the path used to resolve function references and data type references that are used in dynamic SQL statements. It contains a list of one or more schema-names, where schema-names are enclosed in double quotes and separated by commas.

The default value is "SYSIBM","SYSFUN",X where X is the value of the USER special register delimited by double quotes. The schema SYSIBM does not need to be specified. If it is not included in the function path, then it is implicitly assumed as the first schema.

This keyword is used as part of the process for resolving unqualified function and stored procedure references that may have been defined in a schema name other than the current user's schema. The order of the schema names determines the order in which the function and procedure names will be resolved.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Schemas” in *SQL Reference, Volume 1*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “USER special register” in *SQL Reference, Volume 1*

CurrentImplicitXMLParseOption CLI/ODBC configuration keyword

Keyword description:

Sets the value of the CURRENT IMPLICIT XMLPARSE OPTION special register.

db2cli.ini keyword syntax:

CurrentImplicitXMLParseOption = 'STRIP WHITESPACE' | 'PRESERVE WHITESPACE'

Default setting:

Whitespace is stripped during implicit non-validating parsing.

Equivalent connection attribute:

SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION

Usage notes:

Setting this keyword issues the SET CURRENT IMPLICIT XMLPARSE OPTION statement after every connection to a database. By default, this statement is not issued.

The SET CURRENT IMPLICIT XMLPARSE OPTION statement sets the CURRENT IMPLICIT XMLPARSE OPTION special register, which controls whether white space is stripped or preserved during implicit non-validating parsing.

CurrentImplicitXMLParseOption does not affect explicit parsing with the XMLPARSE function.

The supported settings for CurrentImplicitXMLParseOption are:

- STRIP WHITESPACE - white space is removed when an XML document is implicitly parsed. This is the default setting.
- PRESERVE WHITESPACE - white space is preserved when an XML document is implicitly parsed.

Related concepts:

- "db2cli.ini initialization file" on page 277

Related reference:

- "Connection attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*
- "CURRENT IMPLICIT XMLPARSE OPTION special register" in *SQL Reference, Volume 1*
- "SET CURRENT IMPLICIT XMLPARSE OPTION statement" in *SQL Reference, Volume 2*
- "XMLPARSE scalar function" in *SQL Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" on page 279

CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword

Keyword description:

Set the value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.

db2cli.ini keyword syntax:

CurrentMaintainedTableTypesForOpt = ALL | FEDERATED_TOOL | NONE | SYSTEM | USER | <list>

Default setting:

System-maintained refresh-deferred materialized query tables are considered in the optimization of a query.

Usage notes:

This keyword defines the default value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. The value of the special register affects the types of tables which are considered in the optimization of a query. Refer to the SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION SQL statement for details on the supported settings of ALL, FEDERATED_TOOL, NONE, SYSTEM, or USER. The <list> option represents a combination of the supported settings, however, ALL and NONE cannot be specified with any other value, and the same value cannot be specified more than once. Separate each value in the list with a comma, for example:

```
CurrentMaintainedTableTypesForOpt = SYSTEM,USER
```

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “dft_mttb_types - Default maintained table types for optimization configuration parameter” in *Performance Guide*
- “SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION statement” in *SQL Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

CurrentPackagePath CLI/ODBC configuration keyword

Keyword description:

Issue 'SET CURRENT PACKAGE PATH = *schema1, schema2, ...*' after every connection.

db2cli.ini keyword syntax:

```
CurrentPackagePath = schema1, schema2, ...
```

Default setting:

The clause is not appended.

Equivalent connection attribute:

```
SQL_ATTR_CURRENT_PACKAGE_PATH
```

Usage notes:

When set, this option issues the command "SET CURRENT PACKAGE PATH = *schema1, schema2, ...*" after every connection to the database. This setting specifies the list of schema names (collection identifiers) that will be searched when there is a package from a different schema.

This keyword is best suited for use with ODBC static processing applications, rather than CLI applications.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

CurrentPackageSet CLI/ODBC configuration keyword

Keyword description:

Issues the SET CURRENT PACKAGESET statement after every connection.

db2cli.ini keyword syntax:

CurrentPackageSet = *schema name*

Default setting:

The clause is not appended.

Equivalent connection attribute:

SQL_ATTR_CURRENT_PACKAGE_SET

Usage notes:

This option issues the SET CURRENT PACKAGESET SQL statement with the CurrentPackageSet value after every connection to a database. By default this clause is not appended.

The SET CURRENT PACKAGESET SQL statement sets the schema name (collection identifier) that is used to select the package to use for subsequent SQL statements.

CLI/ODBC applications issue dynamic SQL statements. Using this option you can control the privileges used to run these statements:

- Choose a schema to use when running SQL statements from CLI/ODBC applications.
- Ensure the objects in the schema have the desired privileges and then rebind accordingly.
- Set the CurrentPackageSet option to this schema.

The SQL statements from the CLI/ODBC applications will now run under the specified schema and use the privileges defined there.

The following package set names are reserved: NULLID, NULLIDR1, NULLIDRA.

If both the Reopt and CurrentPackageSet keywords are specified, CurrentPackageSet takes precedence.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “SET CURRENT PACKAGESET statement” in *SQL Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Reopt CLI/ODBC configuration keyword” on page 360

CurrentRefreshAge CLI/ODBC configuration keyword

Keyword description:

Set the value of the CURRENT REFRESH AGE special register.

db2cli.ini keyword syntax:

CurrentRefreshAge = 0 | ANY | **positive integer**

Default setting:

Only materialized query tables defined with REFRESH IMMEDIATE may be used to optimize the processing of a query.

Usage notes:

Setting this keyword sets the value of the CURRENT REFRESH AGE special register.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CURRENT REFRESH AGE special register” in *SQL Reference, Volume 1*
- “SET CURRENT REFRESH AGE statement” in *SQL Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

CurrentSchema CLI/ODBC configuration keyword

Keyword description:

Specify the schema used in a SET CURRENT SCHEMA statement upon a successful connection.

db2cli.ini keyword syntax:

CurrentSchema = *schema name*

Default setting:

No statement is issued.

Usage notes:

Upon a successful connect, if this option is set, a SET CURRENT SCHEMA statement is sent to the DBMS. This allows the end user or application to name SQL objects without having to qualify them by schema name.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SET SCHEMA statement” in *SQL Reference, Volume 2*

CurrentSQLID CLI/ODBC configuration keyword

Keyword description:

Specify the ID used in a SET CURRENT SQLID statement sent to the DBMS upon a successful connection.

db2cli.ini keyword syntax:

CurrentSQLID = *current_sqlid*

Default setting:

No statement is issued.

Only applicable when:

connecting to those DB2 DBMS's where SET CURRENT SQLID is supported.

Usage notes:

Upon a successful connection, if this option is set, a SET CURRENT SQLID statement is sent to the DBMS. This allows the end user and the application to name SQL objects without having to qualify them by schema name.

Related concepts:

- "db2cli.ini initialization file" on page 277

Related reference:

- "CLI/ODBC configuration keywords listing by category" on page 279
- "SET SCHEMA statement" in *SQL Reference, Volume 2*

CursorHold CLI/ODBC configuration keyword

Keyword description:

Effect of a transaction completion on open cursors.

db2cli.ini keyword syntax:

CursorHold = 1 | 0

Default setting:

Selected--Cursors are not destroyed.

Equivalent statement attribute:

SQL_ATTR_CURSOR_HOLD

Usage notes:

This option controls the effect of a transaction completion on open cursors.

1 = SQL_CURSOR_HOLD_ON, the cursors are not destroyed when the transaction is committed (default).

0 = SQL_CURSOR_HOLD_OFF, the cursors are destroyed when the transaction is committed.

Note: Cursors are always closed when transactions are rolled back.

This option affects the result returned by SQLGetInfo() when called with SQL_CURSOR_COMMIT_BEHAVIOR or SQL_CURSOR_ROLLBACK_BEHAVIOR. The value of CursorHold is ignored if connecting to DB2 Server for VSE & VM where cursor with hold is not supported.

You can use this option to tune performance. It can be set to SQL_CURSOR_HOLD_OFF (0) if you are sure that your application:

1. Does not have behavior that is dependent on the SQL_CURSOR_COMMIT_BEHAVIOR or the SQL_CURSOR_ROLLBACK_BEHAVIOR information returned via SQLGetInfo(), and
2. Does not require cursors to be preserved from one transaction to the next.

The DBMS will operate more efficiently with CursorHold disabled, as resources no longer need to be maintained after the end of a transaction.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLGetInfo function (CLI) - Get general information” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

CursorTypes CLI/ODBC configuration keyword

Keyword description:

Specify which cursor types are permitted.

db2cli.ini keyword syntax:

CursorTypes = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

Default setting:

Forward-only, static, keyset-driven, and dynamic cursors are supported if the server supports them.

Usage notes:

The CursorTypes keyword is a bitmask that indicates what types of cursors an application can open:

- 0x0 - forward-only (can always be opened)
- 0x1 - static
- 0x2 - keyset-driven
- 0x4 - dynamic

For example,

- to prevent applications from opening dynamic scrollable cursors, set CursorTypes to 3.
- to allow applications to open only non-scrollable cursors, set CursorTypes to 0.

This keyword only affects calls made to the following DB2 CLI functions:

- SQLBulkOperations()
- SQLExecDirect()
- SQLExecute()
- SQLFetchScroll()
- SQLPrepare()
- SQLSetPos()

Related concepts:

- “Cursor considerations for CLI applications” on page 68
- “Cursors in CLI applications” on page 65
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecDirect function (CLI) - Execute a statement directly” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLExecute function (CLI) - Execute a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPrepare function (CLI) - Prepare a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSetPos function (CLI) - Set the cursor position in a rowset” in *Call Level Interface Guide and Reference, Volume 2*

Database CLI/ODBC configuration keyword

Keyword description:

Database on the server to connect to when using a File DSN.

db2cli.ini keyword syntax:

Database = *database name*

Default setting:

None

Only applicable when:

Protocol set to TCPIP

Usage notes:

When using a File DSN you must use this option to specify the database on the server to connect to. This value has nothing to do with any database alias name specified on the client, it must be set to the database name on the server itself.

This setting is only considered when the Protocol option is set to TCPIP.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Hostname CLI/ODBC configuration keyword” on page 327
- “Protocol CLI/ODBC configuration keyword” on page 356
- “ServiceName CLI/ODBC configuration keyword” on page 366

DateTimeStringFormat CLI/ODBC configuration keyword

Keyword description:

Specify the format to use when inserting date or time data into character columns.

db2cli.ini keyword syntax:

DateTimeStringFormat = JIS | ISO | EUR | USA

Default setting:

The JIS format is used when date or time data is inserted into character columns.

Usage notes:

The DateTimeStringFormat keyword controls the format in which date or time data is inserted into character columns. This setting affects the insertion of SQL_C_DATE, SQL_C_TIME, or SQL_C_TIMESTAMP data into the following column types:

- SQL_CHAR
- SQL_VARCHAR
- SQL_LONGVARCHAR
- SQL_CLOB

This keyword also affects the format of date or time columns that are retrieved into character strings. For example, retrieving data from an SQL_TIMESTAMP column into an SQL_C_CHAR string will be affected by the setting of this keyword.

The four setting values are as follows:

Format	Date	Time	Timestamp
JIS	yyyy-mm-dd	hh:mm:ss	yyyy-mm-dd hh:mm:ss.ffffff
ISO	yyyy-mm-dd	hh.mm.ss	yyyy-mm-dd- hh.mm.ss.ffffff
EUR	dd.mm.yyyy	hh.mm.ss	yyyy-mm-dd hh:mm:ss.ffffff*
USA	mm/dd/yyyy	hh:mm AM or PM	yyyy-mm-dd hh:mm:ss.ffffff*

*Timestamps will take the default format if EUR or USA is specified. The default format is JIS.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “Datetime values” in *SQL Reference, Volume 1*
- “CLI/ODBC configuration keywords listing by category” on page 279

DB2Degree CLI/ODBC configuration keyword

Keyword description:

Set the degree of parallelism for the execution of SQL statements.

db2cli.ini keyword syntax:

DB2Degree = 0 | integer value from 1 to 32767 | ANY

Default setting:

No SET CURRENT DEGREE statement is issued.

Only applicable when:

connecting to a cluster database system.

Usage notes:

If the value specified is anything other than 0 (the default) then DB2 CLI will issue the following SQL statement after a successful connection:

```
SET CURRENT DEGREE value
```

This specifies the degree of parallelism for the execution of the SQL statements. The database manager will determine the degree of parallelism if you specify ANY.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SET CURRENT DEGREE statement” in *SQL Reference, Volume 2*

DB2Explain CLI/ODBC configuration keyword

Keyword description:

Determines whether Explain snapshot and/or Explain table information will be generated by the server.

db2cli.ini keyword syntax:

DB2Explain = 0 | 1 | 2 | 3

Default setting:

Neither Explain snapshot nor Explain table information will be generated by the server.

Equivalent connection attribute:

SQL_ATTR_DB2EXPLAIN

Usage notes:

This keyword determines whether Explain snapshot and/or Explain table information will be generated by the server.

- 0 = both off (default)

A ‘SET CURRENT EXPLAIN SNAPSHOT=NO’ and a ‘SET CURRENT EXPLAIN MODE=NO’ statement will be sent to the server to disable both the Explain snapshot and the Explain table information capture facilities.

- 1 = Only Explain snapshot facility on

A 'SET CURRENT EXPLAIN SNAPSHOT=YES' and a 'SET CURRENT EXPLAIN MODE=NO' statement will be sent to the server to enable the Explain snapshot facility, and disable the Explain table information capture facility.

- 2 = Only Explain table information capture facility on

A 'SET CURRENT EXPLAIN MODE=YES' and a 'SET CURRENT EXPLAIN SNAPSHOT=NO' will be sent to the server to enable the Explain table information capture facility and disable the Explain snapshot facility.

- 3 = Both on

A 'SET CURRENT EXPLAIN MODE=YES' and a 'SET CURRENT EXPLAIN SNAPSHOT=YES' will be sent to the server to enable both the Explain snapshot and the Explain table information capture facilities.

Explain information is inserted into Explain tables, which must be created before the Explain information can be generated. The current authorization ID must have INSERT privilege for the Explain tables.

Related concepts:

- "db2cli.ini initialization file" on page 277

Related reference:

- "CLI and ODBC function summary" in *Call Level Interface Guide and Reference, Volume 2*
- "Connection attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*
- "EXPLAIN statement" in *SQL Reference, Volume 2*

DB2NETNamedParam CLI/ODBC configuration keyword

Keyword description:

Specify if named parameters are used by DB2 .NET applications.

db2cli.ini keyword syntax:

DB2NETNamedParam = 0 | 1

Default setting:

The DB2 .NET Data Provider recognizes named parameters as parameters, but ignores positioned parameters, in SQL statements.

Usage notes:

By default, the DB2 .NET Data Provider processes tokens in an SQL statement with the format "@<paramname>" as named parameters and ignores any positioned parameters, where positioned parameters are specified with a '?' character.

The following is an example of a query that contains a named parameter:

```
SELECT * FROM T1 WHERE C1 = @param1
```

This is an example of a query that contains a positioned parameter:

```
SELECT * FROM T1 WHERE C1 = ?
```

Specify 0 to indicate that only positioned parameters will be recognized as parameters in SQL statements. This setting can improve application performance by reducing the overhead required to process named parameters.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

DB2Optimization CLI/ODBC configuration keyword

Keyword description:

Set the query optimization level.

db2cli.ini keyword syntax:

DB2Optimization = *integer value from 0 to 9*

Default setting:

No SET CURRENT QUERY OPTIMIZATION statement issued.

Usage notes:

If this option is set then DB2 CLI will issue the following SQL statement after a successful connection:

```
SET CURRENT QUERY OPTIMIZATION positive number
```

This specifies the query optimization level at which the optimizer should operate the SQL queries.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SET CURRENT QUERY OPTIMIZATION statement” in *SQL Reference, Volume 2*

DBAlias CLI/ODBC configuration keyword

Keyword description:

Specify the database alias for a Data Source Name greater than 8 characters.

db2cli.ini keyword syntax:

DBAlias = *dbalias*

Default setting:

Use the DB2 database alias as the ODBC Data Source Name.

Usage notes:

This keyword allows for Data Source Names of greater than 8 single byte characters. The Data Source Name (DSN) is the name, enclosed in square brackets, that denotes the section header in the db2cli.ini. Typically, this section header is the database alias name which has a maximum length of 8 bytes. A user who wishes to refer to the data source with a longer, more meaningful name, can place the longer name in the section header, and set this keyword value to the database alias used on the CATALOG command. Here is an example:

```
; The much longer name maps to an 8 single byte character dbalias  
[MyMeaningfulName]  
DBAlias=DB2DBT10
```

The end user can specify [MyMeaningfulName] as the name of the data source on connect while the actual database alias is DB2DBT10.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CATALOG DATABASE command” in *Command Reference*
- “CLI/ODBC configuration keywords listing by category” on page 279

DBName CLI/ODBC configuration keyword

Keyword description:

Specify the database name to reduce the time it takes for the application to query z/OS or OS/390 table information.

db2cli.ini keyword syntax:

DBName = *dbname*

Default setting:

Do not filter on the DBNAME column.

Only applicable when:

connecting to DB2 for z/OS and OS/390.

Usage notes:

This option is only used when connecting to DB2 for z/OS and OS/390, and only if (*base*) table catalog information is requested by the application. If a large number of tables exist in the z/OS or OS/390 subsystem, a *dbname* can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application.

If this option is set then the statement IN DATABASE *dbname* will be appended to various statements such as CREATE TABLE.

This value maps to the DBNAME column in the z/OS or OS/390 system catalog tables. If no value is specified, or if views, synonyms, system tables, or aliases are also specified via TableType, only table information will be restricted; views, aliases, and synonyms are not restricted with DBName. It can be used in conjunction with SchemaList, and TableType to further limit the number of tables for which information will be returned.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SchemaList CLI/ODBC configuration keyword” on page 365
- “TableType CLI/ODBC configuration keyword” on page 374

DecimalFloatRoundingMode CLI/ODBC configuration keyword

Keyword description:

Sets the default rounding mode when working with servers that support the DECIMAL FLOAT SQL type. Currently only z/OS v9 has support for this.

db2cli.ini keyword syntax:

DecimalFloatRoundingMode = 0 | 1 | 2 | 3 | 4 | 5 | 6

Default setting:

0 (Half even rounding mode)

Equivalent connection attribute:

SQL_ATTR_DECFLOAT_ROUNDING_MODE

Usage notes:

The decimal float rounding mode determines what type of rounding will be used if a value is put into a decimal float variable or column but the value has more digits than are allowed in the decimal float. This can occur when inserting, updating, converting from another type, or as the result of a mathematical operation.

The value of this keyword determines the decimal float rounding mode that will be used for new connections unless another mode is specified by a connection attribute for that connection. For any given connection both DB2 CLI and DB2 will use the same decimal float rounding mode for all action initiated as part of that connection.

The settings correspond to these decimal float rounding modes:

- 0 = Half even (default)
- 1 = Half up
- 2 = Down
- 3 = Ceiling
- 4 = Floor
- 5 = Half down
- 6 = Up

The different modes are:

Half even (default)

In this mode DB2 CLI and DB2 use the number that will fit in the target variable and that is closest to the original value. If two numbers are equally close, they use the one that is even. This mode produces the smallest rounding errors over large amounts of data.

Half up

In this mode DB2 CLI and DB2 use the number that will fit in the target variable and that is closest to the original value. If two numbers are equally close, they use the one that is greater than the original value.

Half down

In this mode DB2 CLI and DB2 use as much of the original value as will fit in the target variable. They ignore any leftover digits. This is often called truncation.

Ceiling

In this mode DB2 CLI and DB2 use the smallest number that will fit in the target variable and that is greater than or equal to the original value.

Floor In this mode DB2 CLI and DB2 use the largest number that will fit in the target variable and that is less than or equal to the original value.

Up In this mode DB2 CLI and DB2 use the number that will fit in the target variable and that is closest to the original value and for which the absolute value is not less than the absolute value of the original value. You can also think of this as rounding away from zero or as using floor for negative values and using ceiling for positive values.

Down In this mode DB2 CLI and DB2 use the number that will fit in the target variable and that is closest to the original value and for which the absolute value is not greater than the absolute value of the original value. You can also think of this as rounding toward zero or as using ceiling for negative values and using floor for positive values.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

DeferredPrepare CLI/ODBC configuration keyword

Keyword description:

Minimize network flow by combining the PREPARE request with the corresponding execute request.

db2cli.ini keyword syntax:

DeferredPrepare = 0 | 1

Default setting:

The prepare request will be delayed until the execute request is sent.

Equivalent statement attribute:

SQL_ATTR_DEFERRED_PREPARE

Usage notes:

Defers sending the PREPARE request until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance.

- 0 = SQL_DEFERRED_PREPARE_OFF. The PREPARE request will be executed the moment it is issued.
- 1 = SQL_DEFERRED_PREPARE_ON (default). Defer the execution of the PREPARE request until the corresponding execute request is issued.

If the target DBMS does not support deferred prepare, the client disables deferred prepare for that connection.

Note: When deferred prepare is enabled, the row and cost estimates normally returned in the SQLERRD(3) and SQLERRD(4) of the SQLCA of a PREPARE statement may become zeros. This may be of concern to users who want to use these values to decide whether or not to continue the SQL statement.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Deferred prepare in CLI applications” on page 26

Related tasks:

- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “PREPARE statement” in *SQL Reference, Volume 2*
- “SQLCA (SQL communications area)” in *SQL Reference, Volume 1*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

DescribeCall CLI/ODBC configuration keyword

Keyword description:

Determine when stored procedure arguments are described.

db2cli.ini keyword syntax:

DescribeCall = 1 | -1

Default setting:

DB2 CLI does not request stored procedure argument describe information when it prepares a CALL statement.

Equivalent connection attribute:

SQL_ATTR_DESCRIBE_CALL

Usage notes:

By default, DB2 CLI does not request input parameter describe information when it prepares a CALL statement. If an application has correctly bound parameters to a statement, then this describe information is unnecessary and not requesting it improves performance.

The option values are:

- 1 = SQL_DESCRIBE_CALL_BEFORE. DB2 CLI always requests describe information from the server, ignoring the binding information provided by the application. Setting DescribeCall to 1 will also set DeferredPrepare to 0 which means that describe information will also be requested for dynamic SQL statements. Note that setting DeferredPrepare to 0 will not set DescribeCall to 1.
- -1 = SQL_DESCRIBE_CALL_DEFAULT (default). DB2 CLI does not request describe information from the server and uses the binding information provided by the application. If the CALL statement execution fails, then the DB2 CLI error recovery logic requests input parameter describe information from the server and issues the CALL statement again.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “DeferredPrepare CLI/ODBC configuration keyword” on page 316

DescribeInputOnPrepare CLI/ODBC configuration keyword

Keyword description:

Enable or disable the request for describe information when an SQL statement is prepared.

db2cli.ini keyword syntax:

DescribeInputOnPrepare = 0 | 1

Default setting:

Do not request describe information when preparing an SQL statement.

Usage notes:

By default, DB2 CLI does not request input parameter describe information when it prepares an SQL statement. If an application has correctly bound parameters to a statement, then this describe information is unnecessary and not requesting it improves performance. If, however, parameters have not been correctly bound, then statement execution will fail and cause the CLI error recovery retry logic to request input parameter describe information. The result is an additional server request and reduced performance, compared to if the describe information had been requested with the prepare. Setting DescribeInputOnPrepare to 1 causes the input describe information to be requested with the prepare. This setting may improve performance for applications which rely heavily on the CLI retry logic to recover from application binding errors.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related tasks:

- “Binding parameter markers in CLI applications” on page 29
- “Preparing and executing SQL statements in CLI applications” on page 24

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

DescribeOutputLevel CLI/ODBC configuration keyword

Keyword description:

Set the level of output column describe information that is requested by the CLI driver during prepare or describe requests.

db2cli.ini keyword syntax:

DescribeOutputLevel = 0 | 1 | 2 | 3

Default setting:

Request the describe information listed in level 2 of Table 22 on page 320.

Equivalent connection attribute:

SQL_ATTR_DESCRIBE_OUTPUT_LEVEL

Usage notes:

This keyword controls the amount of information the CLI driver requests on a prepare or describe request. By default, when the server receives a describe request, it returns the information contained in level 2 of Table 22 on page 320 for the result set columns. An application, however, might not need all of this information or might need additional information. Setting the DescribeOutputLevel keyword to a level that suits the needs of the client application might improve performance because the describe data transferred between the client and server is limited to the minimum amount that the application requires. If the DescribeOutputLevel setting is set too low, it might impact the functionality of the application (depending on the application's requirements). The CLI functions to retrieve the describe information might not fail in this case, but the information returned might be incomplete. Supported settings for DescribeOutputLevel are:

- 0 - no describe information is returned to the client application
- 1 - describe information categorized in level 1 (see Table 22 on page 320) is returned to the client application
- 2 - (default) describe information categorized in level 2 (see Table 22 on page 320) is returned to the client application
- 3 - describe information categorized in level 3 (see Table 22 on page 320) is returned to the client application

The following table lists the fields that form the describe information that the server returns when it receives a prepare or describe request. These fields are grouped into levels, and the DescribeOutputLevel CLI/ODBC configuration keyword controls which levels of describe information the CLI driver requests.

Notes:

1. Not all levels of describe information are supported by all DB2 servers. All levels of describe information are supported on the following DB2 servers: DB2 for Linux, UNIX, and Windows Version 8 and later, DB2 for z/OS Version 8 and later, and DB2 for iSeries Version 5 Release 3 and later. All other DB2 servers support only the 2 or 0 setting for DescribeOutputLevel.
2. The default behavior allows CLI to promote the level to 3 if the application asks for describe information that was not initially retrieved using the default level 2. This might result in two network flows to the server. If an application uses this keyword to explicitly set a describe level, then no promotion will occur. Therefore, if the keyword is used to set the describe level to 2, CLI will not promote to level 3 even if the application asks for extended information.

Table 22. Levels of describe information

Level 1	Level 2	Level 3
SQL_DESC_COUNT	all fields of level 1 and: SQL_DESC_NAME SQL_DESC_LABEL SQL_COLUMN_NAME SQL_DESC_UNNAMED SQL_DESC_TYPE_NAME SQL_DESC_DISTINCT_TYPE SQL_DESC_REFERENCE_TYPE SQL_DESC_STRUCTURED_TYPE SQL_DESC_USER_TYPE SQL_DESC_LOCAL_TYPE_NAME SQL_DESC_USER_DEFINED_ TYPE_CODE	all fields of levels 1 and 2 and: SQL_DESC_BASE_COLUMN_NAME SQL_DESC_UPDATABLE SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_SCHEMA_NAME SQL_DESC_CATALOG_NAME SQL_DESC_TABLE_NAME SQL_DESC_BASE_TABLE_NAME
SQL_COLUMN_COUNT		
SQL_DESC_TYPE		
SQL_DESC_CONCISE_TYPE		
SQL_COLUMN_LENGTH		
SQL_DESC_OCTET_LENGTH		
SQL_DESC_LENGTH		
SQL_DESC_PRECISION		
SQL_COLUMN_PRECISION		
SQL_DESC_SCALE		
SQL_COLUMN_SCALE		
SQL_DESC_DISPLAY_SIZE		
SQL_DESC_NULLABLE		
SQL_COLUMN_NULLABLE		
SQL_DESC_UNSIGNED		
SQL_DESC_SEARCHABLE		
SQL_DESC_LITERAL_SUFFIX		
SQL_DESC_LITERAL_PREFIX		
SQL_DESC_CASE_SENSITIVE		
SQL_DESC_FIXED_PREC_SCALE		

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

DescribeParam CLI/ODBC configuration keyword

Keyword description:

Specify if SQLDescribeParam() is supported.

db2cli.ini keyword syntax:

DescribeParam = 0 | 1

Default setting:

The SQLDescribeParam() function is enabled.

Usage notes:

This keyword enables or disables the SQLDescribeParam() function.

When set to 1 (default), SQLDescribeParam() is enabled and SQLGetFunctions() will return SQLDescribeParam() as supported.

When set to 0, SQLDescribeParam() is disabled. If SQLDescribeParam() is called, CLI0150E “Driver not capable” will be returned. SQLGetFunctions() will return SQLDescribeParam() as not supported.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “SQLDescribeParam function (CLI) - Return description of a parameter marker” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetFunctions function (CLI) - Get functions” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

DiagLevel CLI/ODBC configuration keyword

Keyword description:

Indicates the diagnostic level.

db2cli.ini keyword syntax:

DiagLevel = 0 | 1 | 2 | 3 | 4

Default setting:

3

Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is applicable only at Environment Handle allocation time for an entire process.

This is equivalent to the database manager parameter DIAGLEVEL. See diaglevel - Diagnostic error capture level configuration parameter for a details about the values.

DiagPath CLI/ODBC configuration keyword

Keyword description:

A string value indicating the path of the db2diag.log file.

db2cli.ini keyword syntax:

DiagPath = *existing directory*

Default setting:

The default value is the db2dump directory on UNIX and Linux operating systems, and the db2 directory on Windows operating systems.

Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is equivalent to the database manager parameter DIAGPATH.

DisableKeysetCursor CLI/ODBC configuration keyword

Keyword description:

Disables keyset-driven scrollable cursors.

db2cli.ini keyword syntax:

DisableKeysetCursor = 0 | 1

Default setting:

Keyset-driven scrollable cursors are returned when requested.

Usage notes:

When set to 1, this keyword forces the CLI driver to return a static cursor to the application, even if the application has requested a keyset-driven scrollable cursor.

The default setting (0) causes keyset-driven cursors to be returned when the application requests them. This keyword can be used to restore behavior before scrollable cursors were supported.

Related concepts:

- “Cursors in CLI applications” on page 65
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

DisableMultiThread CLI/ODBC configuration keyword

Keyword description:

Disable multithreading.

db2cli.ini keyword syntax:

DisableMultiThread = 0 | 1

Default setting:

Multithreading is enabled.

Usage notes:

The CLI/ODBC driver is capable of supporting multiple concurrent threads.

This option is used to enable or disable multi-thread support.

- 0 = Multithreading is enabled (default).
- 1 = Disable multithreading.

If multithreading is disabled then all calls for all threads will be serialized at the process level. Use this setting for multithreaded applications that require serialized behavior.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Multithreaded CLI applications” on page 133

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

DisableUnicode CLI/ODBC configuration keyword

Keyword description:

Disable underlying Unicode support.

db2cli.ini keyword syntax:

DisableUnicode = <not set> | 0 | 1

Default setting:

Unicode support is enabled.

Usage notes:

With Unicode support enabled, and when called by a Unicode application, CLI will attempt to connect to the database using the best client code page possible to ensure there is no unnecessary data loss due to code page conversion. This may increase the connection time as code pages are exchanged, or may cause code page conversions on the client that did not occur before this support was added.

If an application is Unicode (the `SQL_ATTR_ANSI_APP` connection attribute is set to `SQL_AA_FALSE`, or the connection occurred with `SQLConnectW()`), then the `DisableUnicode` keyword can be used to effect three different connection behaviors:

- `DisableUnicode` is not set in the `db2cli.ini` file: If the target database supports Unicode, DB2 CLI will connect in Unicode code pages (1208 and 1200). Otherwise, DB2 CLI will connect in the application code page.
- `DisableUnicode=0` is set: DB2 CLI always connects in Unicode, whether or not the target database supports Unicode.
- `DisableUnicode=1` is set: DB2 CLI always connects in the application code page, whether or not the target database supports Unicode.

Related concepts:

- “`db2cli.ini` initialization file” on page 277
- “Unicode CLI applications” on page 143

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

DSN CLI/ODBC configuration keyword

Keyword description:

Name of a data source as returned by `SQLDataSources` or the data sources dialog box of `SQLDriverConnect`.

`db2cli.ini` keyword syntax:

You can not set this keyword in the `db2cli.ini` file.

You can specify the value of this keyword in the connection string in `SQLDriverConnect` like this:

`DSN = database name`

FileDSN CLI/ODBC configuration keyword

Keyword description:

Name of a DSN file from which a connection string will be built for the data source.

`db2cli.ini` keyword syntax:

You can not set this keyword in the `db2cli.ini` file.

You can specify the value of this keyword in the connection string in `SQLDriverConnect` like this:

`FileDSN = file name`

FloatPrecRadix CLI/ODBC configuration keyword

Keyword description:

Force the NUM_PREC_RADIX value of a floating point type to be 2 or 10.

db2cli.ini keyword syntax:

FloatPrecRadix = 2 | 10

Default setting:

Report the NUM_PREC_RADIX as 2 for floating point types, as they have a base of 2, not 10.

Usage notes:

The NUM_PREC_RADIX value represents a data type's base. Binary numbers, such as floating point numbers, have a base of 2, and integers have a base of 10. An application may expect all values in the COLUMN_SIZE field to represent the maximum number of digits, which assumes a NUM_PREC_RADIX value of 10. However, for floating point numeric types, the NUM_PREC_RADIX is 2, in which case the COLUMN_SIZE will report the number of bits in the data type's representation, rather than the maximum number of digits.

FloatPrecRadix can force the NUM_PREC_RADIX to be reported as 10 for floating point data types, in which case the COLUMN_SIZE will report the maximum number of digits.

The FloatPrecRadix keyword affects SQLColumns(), SQLGetDescField() (for the SQL_DESC_NUM_PREC_RADIX field), SQLGetTypeInfo(), SQLProcedureColumns(), and SQLSpecialColumns().

Related concepts:

- "db2cli.ini initialization file" on page 277

Related reference:

- "CLI/ODBC configuration keywords listing by category" on page 279
- "Descriptor FieldIdentifier argument values (CLI)" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLColumns function (CLI) - Get column information for a table" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLGetTypeInfo function (CLI) - Get data type information" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure" in *Call Level Interface Guide and Reference, Volume 2*
- "SQLSpecialColumns function (CLI) - Get special (row identifier) columns" in *Call Level Interface Guide and Reference, Volume 2*

GranteeList CLI/ODBC configuration keyword

Keyword description:

Reduce the amount of information returned when the application gets a list of table or column privileges.

db2cli.ini keyword syntax:

```
GranteeList = " 'userID1', 'userID2',... 'userIDn' "
```

Default setting:

Do not filter the results.

Usage notes:

This option can be used to reduce the amount of information returned when the application gets a list of privileges for tables in a database, or columns in a table. The list of authorization IDs specified is used as a filter; the only tables or columns that are returned are those with privileges that have been granted *TO* those IDs.

Set this option to a list of one or more authorization IDs that have been granted privileges, delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
GranteeList=" 'USER1', 'USER2', 'USER8' "
```

In the above example, if the application gets a list of privileges for a specific table, only those columns that have a privilege granted *TO* USER1, USER2, or USER8 would be returned.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “GrantorList CLI/ODBC configuration keyword” on page 325

GrantorList CLI/ODBC configuration keyword

Keyword description:

Reduce the amount of information returned when the application gets a list of table or column privileges.

db2cli.ini keyword syntax:

```
GrantorList = " 'userID1', 'userID2',... 'userIDn' "
```

Default setting:

Do not filter the results.

Usage notes:

This option can be used to reduce the amount of information returned when the application gets a list of privileges for tables in a database, or columns in a table. The list of authorization IDs specified is used as a filter; the only tables or columns that are returned are those with privileges that have been granted *BY* those IDs.

Set this option to a list of one or more authorization IDs that have granted privileges, delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
GrantorList=" 'USER1', 'USER2', 'USER8' "
```

In the above example, if the application gets a list of privileges for a specific table, only those columns that have a privilege granted *BY* USER1, USER2, or USER8 would be returned.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “GranteeList CLI/ODBC configuration keyword” on page 324

Graphic CLI/ODBC configuration keyword

Keyword description:

Specifies if DB2 CLI returns SQL_GRAPHIC (double-byte character) as a supported SQL data type and what unit is used to report GRAPHIC column length.

db2cli.ini keyword syntax:

Graphic = 0 | 1 | 2 | 3

Default setting:

The SQL_GRAPHIC data type is not returned as a supported SQL data type, and the length of GRAPHIC columns equals the maximum number of DBCS characters in the column.

Usage Notes:

The Graphic keyword controls whether the SQL_GRAPHIC (double-byte character) data type is reported as a supported SQL data type when SQLGetTypeInfo() is called, as well as what unit is used to report the length of GRAPHIC columns for all DB2 CLI functions that return length or precision as either output arguments or as part of a result set.

Set the Graphic keyword as follows:

- 0 - SQL_GRAPHIC is not returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of DBCS characters in the column.
- 1 - SQL_GRAPHIC is returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of DBCS characters in the column.
- 2 - SQL_GRAPHIC is not returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of bytes in the column.
- 3 - SQL_GRAPHIC is returned as a supported SQL data type, and the reported length of GRAPHIC columns equals the maximum number of bytes in the column.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

- “SQLGetTypeInfo function (CLI) - Get data type information” in *Call Level Interface Guide and Reference, Volume 2*

Hostname CLI/ODBC configuration keyword

Keyword description:

The server system’s host name or IP address, used with file DSN or in a DSN-less connection.

db2cli.ini keyword syntax:

Hostname = *host name* | *IP Address*

Default setting:

None

Only applicable when:

Protocol set to TCPIP

Usage notes:

Use this option in conjunction with the ServiceName option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the Protocol option is set to TCPIP.

Specify either the server system’s host name or its IP address.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Protocol CLI/ODBC configuration keyword” on page 356
- “ServiceName CLI/ODBC configuration keyword” on page 366

IgnoreWarnings CLI/ODBC configuration keyword

Keyword description:

Ignore database manager warnings.

db2cli.ini keyword syntax:

IgnoreWarnings = 0 | 1

Default setting:

Warnings are returned as normal.

Usage notes:

On rare occasions, an application will not correctly handle warning messages. This keyword can be used to indicate that warnings from the database manager are not to be passed to the application. The possible settings are:

- 0 - Warnings are reported as usual (default)
- 1 - Database manager warnings are ignored and SQL_SUCCESS is returned. Warnings from the DB2 CLI/ODBC driver are still returned; many are required for normal operation.

Although this keyword can be used on its own, it can also be used with the WarningList CLI/ODBC configuration keyword.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “IgnoreWarnList CLI/ODBC configuration keyword” on page 328
- “WarningList CLI/ODBC configuration keyword” on page 391

IgnoreWarnList CLI/ODBC configuration keyword

Keyword description:

Ignore specified sqlstates.

db2cli.ini keyword syntax:

IgnoreWarnList = “'sqlstate1', 'sqlstate2', ...”

Default setting:

Warnings are returned as normal

Usage notes:

On rare occasions an application may not correctly handle some warning messages, but does not want to ignore all warning messages. This keyword can be used to indicate which warnings are not to be passed on to the application. The IgnoreWarnings keyword should be used if all database manager warnings are to be ignored.

If an sqlstate is included in both IgnoreWarnList and WarningList, it will be ignored altogether.

Each sqlstate must be in uppercase, delimited with single quotes and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
IgnoreWarnList=" '01000', '01004', '01504' "
```

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “IgnoreWarnings CLI/ODBC configuration keyword” on page 327
- “WarningList CLI/ODBC configuration keyword” on page 391

Instance CLI/ODBC configuration keyword

Keyword description:

Instance name for a local IPC connection for file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

Instance = *instance name*

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this keyword, you must also set the following options:

- Database; and
- Protocol=IPC.

Interrupt CLI/ODBC configuration keyword

Keyword description:

Interrupt processing mode.

db2cli.ini keyword syntax:

Interrupt = 0 | 1 | 2

Default setting:

1

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- Database; and
- Protocol=IPC.

The keyword values have the following meaning:

0 Disables interrupt processing (SQLCancel calls will not interrupt the processing.)

1 Interrupts are supported (default.) In this mode, if the server supports an interrupt, an interrupt will be sent. Otherwise the connection is dropped.

The settings for INTERRUPT_ENABLED (a DB2 Connect gateway setting) and the DB2 registry variable DB2CONNECT_DISCONNECT_ON_INTERRUPT will take precedence over the Interrupt keyword setting of 1.

2 Interrupt drops the connection regardless of server's interrupt capabilities (SQLCancel will drop the connection.)

KeepDynamic CLI/ODBC configuration keyword

Keyword description:

Specify if KEEP_DYNAMIC functionality is available to DB2 CLI applications.

db2cli.ini keyword syntax:

KeepDynamic = 0 | 1

Default setting:

KEEP_DYNAMIC functionality is not available to DB2 CLI applications.

Equivalent connection attribute:

SQL_ATTR_KEEP_DYNAMIC

Usage notes:

The KeepDynamic CLI/ODBC configuration keyword should be set according to how the CLI packages were bound on the DB2 for z/OS and OS/390 server. Set KeepDynamic as follows:

- 0 - if the CLI packages on the server were bound with the KEEP_DYNAMIC NO option
- 1 - if the CLI packages on the server were bound with the KEEP_DYNAMIC YES option

It is recommended that when KeepDynamic is used, the CurrentPackageSet CLI/ODBC keyword also be set. Refer to the documentation about enabling KEEP_DYNAMIC support for details on how these keywords can be used together.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Programming hints and tips for CLI applications” on page 55

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CurrentPackageSet CLI/ODBC configuration keyword” on page 305

KRBPlugin CLI/ODBC configuration keyword

Keyword description:

Kerberos plugin to be used for client side authentication for file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

KRBPlugin = *plugin name*

Default setting:

By default, the value is null on UNIX operating systems, and IBMkrb5 on Windows operating systems.

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies the name of the Kerberos plug-in library to be used for client-side connection authentication. The plug-in is used when the client is authenticated using KERBEROS authentication.

LoadXAInterceptor CLI/ODBC configuration keyword

Keyword description:

Load XA Interceptor for debugging.

db2cli.ini keyword syntax:

LoadXAInterceptor = 0 | 1

Default setting:

The XA Interceptor is not loaded.

Usage notes:

This keyword loads the XA Interceptor for debugging purposes in MTS.

Related concepts:

- “Enablement of COM+ support in C/C++ database applications” in *Developing ADO.NET and OLE DB Applications*
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

LOBCacheSize CLI/ODBC configuration keyword

Keyword description:

Specify maximum cache size (in bytes) for LOBs.

db2cli.ini keyword syntax:

LOBCacheSize = *positive integer*

Default setting:

LOBs are not cached.

Equivalent connection or statement attribute:

SQL_ATTR_LOB_CACHE_SIZE

Usage notes:

The use of LOB locators when retrieving unbound LOB data can be avoided by setting this keyword. For example, if an application does not bind a column prior to calling `SQLFetch()` and then calls `SQLGetData()` to fetch the LOB, if `LOBCacheSize` was set to a value large enough to contain the entire LOB being fetched, then the LOB is retrieved from the LOB cache rather than from a LOB locator. Using the LOB cache instead of the LOB locator in this case improves performance.

Servers that support Dynamic Data Format optimize the return of LOB and XML data depending on the actual length of the data. The LOB and XML data can be returned in its entirety, or as an internal token called a progressive reference. DB2 CLI manages progressive reference data retrieval.

For applications that are querying data on a server that supports Dynamic Data Format, setting the `LOBCacheSize` keyword sets a threshold that is used to determine if the data is returned in its entirety, or as a progressive reference. If the data has a length greater than the `LOBCacheSize` threshold value, the progressive reference will be returned to DB2 CLI to manage, but if the data has a length less than or equal to the `LOBCacheSize` threshold value, the data will be returned in its entirety.

For applications that are querying data on a server that does not support Dynamic Data Format, the `LOBCacheSize` threshold value specifies the maximum defined size of a LOB that DB2 CLI will buffer in memory. If the defined size of a LOB exceeds the value `LOBCacheSize` is set to, then the LOB will not be cached. For example, consider a table that is created with a CLOB column of 100MB currently holding 20MB of data, with `LOBCacheSize` set to 50MB. In this case, even though the size of the LOB itself (20MB) is less than the value set through `LOBCacheSize`, the CLOB column will not be cached because the defined CLOB size (100MB) exceeds the maximum cache size set through `LOBCacheSize` (50MB).

`ClientBuffersUnboundLOBS` is a related keyword.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Large object usage in CLI applications” on page 96
- “LOB locators in CLI applications” on page 98

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “ClientBuffersUnboundLOBS CLI/ODBC configuration keyword” on page 296
- “CLI/ODBC configuration keywords listing by category” on page 279

LOBFileThreshold CLI/ODBC configuration keyword

Keyword description:

Specify maximum number of bytes of LOB data buffered when SQLPutData() is used.

db2cli.ini keyword syntax:

LOBFileThreshold = *positive integer*

Default setting:

25 MB

Usage notes:

This option specifies the maximum number of bytes of LOB data that DB2 CLI will buffer in memory on calls to SQLPutData(). If the specified cache size is exceeded, a temporary file will be created on disk to hold the LOB data before it is sent to the server.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Large object usage in CLI applications” on page 96

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*

LOBMaxColumnSize CLI/ODBC configuration keyword

Keyword description:

Override default COLUMN_SIZE for LOB data types.

db2cli.ini keyword syntax:

LOBMaxColumnSize = *integer greater than zero*

Default setting:

2 Gigabytes (1G for DBCLOB)

Only applicable when:

LongDataCompat or MapXMLDescribe with a LOB type is used.

Usage notes:

This will override the 2 Gigabyte (1G for DBCLOB) value that is returned by SQLGetTypeInfo() for the COLUMN_SIZE column for SQL_CLOB, SQL_BLOB, and SQL_DBCLOB and SQL_XML SQL data types. For SQL_XML, LOBMaxColumnSize must be specified with MapXMLDescribe set to a LOB type. Subsequent CREATE TABLE statements that contain LOB columns will use the column size value you set here instead of the default.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “LongDataCompat CLI/ODBC configuration keyword” on page 333
- “MapXMLDescribe CLI/ODBC configuration keyword” on page 344
- “SQL symbolic and default data types for CLI applications” on page 41

LockTimeout CLI/ODBC configuration keyword

Keyword description:

Set the default value of the LOCKTIMEOUT configuration parameter.

db2cli.ini keyword syntax:

LockTimeout = -1 | 0 | positive integer ≤ 32767

Default setting:

Timeout is turned off (-1), with the application waiting for a lock until either the lock is granted or deadlock occurs.

Usage notes:

The LockTimeout keyword specifies the number of seconds a DB2 CLI application will wait to obtain locks. If the keyword is set to 0, locks will not be waited for. The -1 setting causes the application to wait indefinitely until either the lock is granted or deadlock occurs.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “locktimeout - Lock timeout configuration parameter” in *Performance Guide*
- “CLI/ODBC configuration keywords listing by category” on page 279

LongDataCompat CLI/ODBC configuration keyword

Keyword description:

Report LOBs as long data types or as large object types.

db2cli.ini keyword syntax:

LongDataCompat = 0 | 1 | 2

Default setting:

Reference LOB data types as large object types.

Equivalent connection attribute:

SQL_ATTR_LONGDATA_COMPAT

Usage notes:

This option indicates to DB2 CLI what data type the application expects when working with a database with large object (LOB) columns.

The values for this option are:

- 0 = Reference LOB data types as large object types.
- 1 = Report LOBs as long data types for DB2 CLI/ODBC applications only.
- 2 = Report LOBs as long data types for JDBC applications only. This does not affect applications using the DB2 Universal JDBC Driver.

Table 23. Corresponding large object and long data types for LOB data

Database data type	Large objects (0 - Default)	Long data types (1 — CLI/ODBC; 2 — JDBC)
CLOB	SQL_CLOB	SQL_LONGVARCHAR
BLOB	SQL_BLOB	SQL_LONGVARBINARY
DBCLOB	SQL_DBCLOB	SQL_LONGVARGRAPHIC*

* If the MapGraphicDescribe keyword is set in conjunction with LongDataCompat, DBCLOB columns will return an SQL type of SQL_LONGVARCHAR if MapGraphicDescribe is 1 and SQL_WLONGVARCHAR if MapGraphicDescribe is 2.

This option is useful when running ODBC applications that cannot handle the large object data types.

The DB2 CLI/ODBC option LOBMaxColumnSize can be used in conjunction with this option to reduce the default size declared for the data.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Large object usage in CLI applications” on page 96

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “LOBMaxColumnSize CLI/ODBC configuration keyword” on page 332
- “MapGraphicDescribe CLI/ODBC configuration keyword” on page 339

MapBigintCDefault CLI/ODBC configuration keyword

Keyword description:

Specify the default C type of BIGINT columns and parameter markers.

db2cli.ini keyword syntax:

MapBigintCDefault = 0 | 1 | 2

Default setting:

The default C type representation for BIGINT data is SQL_C_BIGINT.

Usage notes:

MapBigintCDefault controls the C type that is used when SQL_C_DEFAULT is specified for BIGINT columns and parameter markers. This keyword should be

used primarily with Microsoft applications, such as Microsoft Access, which cannot handle 8-byte integers. Set MapBigintCDefault as follows:

- 0 - for the default SQL_C_BIGINT C type representation
- 1 - for an SQL_C_CHAR C type representation
- 2 - for an SQL_C_WCHAR C type representation

This keyword affects the behavior of CLI functions where SQL_C_DEFAULT might be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData()

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*

MapCharToWChar CLI/ODBC configuration keyword

Keyword description:

Specify the default SQL type associated with SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR.

db2cli.ini keyword syntax:

MapCharToWChar = 0 | 1

Default setting:

The default SQL type representation for SQL_CHAR, SQL_VARCHAR and SQL_LONGVARCHAR is used.

Equivalent connection attribute:

SQL_ATTR_MAPCHAR

Usage notes:

MapCharToWChar controls the SQL type that is returned when describing SQL_CHAR, SQL_VARCHAR and SQL_LONGVARCHAR columns or parameter markers.

Set MapCharToWChar as follows:

- 0 - to return the default SQL type representation
- 1 - to return SQL_CHAR as SQL_WCHAR, SQL_VARCHAR as SQL_WVARCHAR, and SQL_LONGVARCHAR as SQL_WLONGVARCHAR

Only the following DB2 CLI functions are affected by setting MapCharToWChar:

- SQLColumns()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “SQLColumns function (CLI) - Get column information for a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeParam function (CLI) - Return description of a parameter marker” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLColAttribute function (CLI) - Return a column attribute” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279

MapDateCDefault CLI/ODBC configuration keyword

Keyword description:

Specify the default C type of DATE columns and parameter markers.

db2cli.ini keyword syntax:

MapDateCDefault = 0 | 1 | 2

Default setting:

The default C type representation for DATE data is SQL_C_TYPE_DATE.

Usage notes:

MapDateCDefault controls the C type that is used when SQL_C_DEFAULT is specified for DATE columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which assume SQL_C_CHAR as the default C type for datetime values. Set MapDateCDefault as follows:

- 0 - for the default SQL_C_TYPE_DATE C type representation: a struct containing numeric members for year, month and day
- 1 - for an SQL_C_CHAR C type representation: "2004-01-01"
- 2 - for an SQL_C_WCHAR C type representation: "2004-01-01" in UTF-16.

This keyword affects the behavior of CLI functions where SQL_C_DEFAULT may be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “MapTimeCDefault CLI/ODBC configuration keyword” on page 340
- “MapTimestampCDefault CLI/ODBC configuration keyword” on page 342
- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*

MapDateDescribe CLI/ODBC configuration keyword

Keyword description:

Controls the SQL data type returned when DATE columns and parameter markers are described.

db2cli.ini keyword syntax:

MapDateDescribe = 0 | 1 | 2

Default setting:

The default SQL data type for DATE data is returned: SQL_DATE for ODBC 2.0 or SQL_TYPE_DATE for ODBC 3.0.

Usage notes:

To control the SQL data type that is returned when DATE columns and parameter markers are described, set MapDateDescribe as follows:

- 0 - to return the default SQL data type: SQL_DATE for ODBC 2.0 or SQL_TYPE_DATE for ODBC 3.0
- 1 - to return the SQL_CHAR SQL data type
- 2 - to return the SQL_WCHAR SQL data type

Only the following DB2 CLI functions are affected by setting MapDateDescribe:

- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “SQLColumns function (CLI) - Get column information for a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeParam function (CLI) - Return description of a parameter marker” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSpecialColumns function (CLI) - Get special (row identifier) columns” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “MapGraphicDescribe CLI/ODBC configuration keyword” on page 339
- “MapTimeDescribe CLI/ODBC configuration keyword” on page 340
- “MapTimestampDescribe CLI/ODBC configuration keyword” on page 342

MapDecimalFloatDescribe CLI/ODBC configuration keyword

Keyword description:

Specify the default C type and reported data type of DECFLOAT columns and parameter markers.

db2cli.ini keyword syntax:

MapDecimalFloatDescribe = 0 | 1 | 2 | 3

Default setting:

0

Usage notes:

MapDecimalFloatDescribe controls the default C type to be used for columns and parameters with a data type of DECFLOAT. It affects the behavior of CLI functions for which SQL_C_DEFAULT can be specified as the C type of a column or parameter. Examples of such functions include SQLBindParameter(), SQLBindCol(), and SQLGetData(),

MapDecimalFloatDescribe also controls the type that will be reported for columns and parameters that have a data type of DECFLOAT. This affects CLI functions that return information about parameters and columns. Examples of such functions include SQLColAttribute() and SQLDescribeParam().

Use this configuration keyword for applications that cannot handle decimal float types or when you would rather always deal with decimal float types as some other type.

Here are the allowed values:

Table 24.

Value	DECFLOAT columns and parameters are reported as being this type	DECFLOAT columns and parameters use this default C type
0	SQL_DECFLOAT	SQL_C_CHAR
1	SQL_VARCHAR	SQL_C_CHAR
2	SQL_WVARCHAR	SQL_C_WCHAR
3	SQL_DOUBLE	SQL_C_DOUBLE

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

MapGraphicDescribe CLI/ODBC configuration keyword

Keyword description:

Controls the SQL data type returned when GRAPHIC, VARGRAPHIC, and LONGVARGRAPHIC columns and parameter markers are described.

db2cli.ini keyword syntax:

MapGraphicDescribe = 0 | 1 | 2

Default setting:

The default SQL data types are returned: SQL_GRAPHIC for GRAPHIC columns, SQL_VARGRAPHIC for VARGRAPHIC columns, and SQL_LONGVARGRAPHIC for LONG VARGRAPHIC columns.

Usage notes:

To control the SQL data type that is returned when GRAPHIC-based columns and parameter markers are described, set MapGraphicDescribe as follows:

- 0 - to return the default SQL data types
- 1 - to return the CHAR-based SQL data types: SQL_CHAR for GRAPHIC columns, SQL_VARCHAR for VARGRAPHIC columns, and SQL_LONGVARCHAR for LONG VARGRAPHIC columns
- 2 - to return the WCHAR-based SQL data types: SQL_WCHAR for GRAPHIC columns, SQL_WVARCHAR for VARGRAPHIC columns, and SQL_WLONGVARCHAR for LONG VARGRAPHIC columns

Only the following DB2 CLI functions are affected by setting MapGraphicDescribe:

- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “MapDateDescribe CLI/ODBC configuration keyword” on page 337
- “MapTimeDescribe CLI/ODBC configuration keyword” on page 340
- “MapTimestampDescribe CLI/ODBC configuration keyword” on page 342
- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeParam function (CLI) - Return description of a parameter marker” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*

- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSpecialColumns function (CLI) - Get special (row identifier) columns” in *Call Level Interface Guide and Reference, Volume 2*

MapTimeCDefault CLI/ODBC configuration keyword

Keyword description:

Specify the default C type of TIME columns and parameter markers.

db2cli.ini keyword syntax:

MapTimeCDefault = 0 | 1 | 2

Default setting:

The default C type representation for TIME data is SQL_C_TYPE_TIME.

Usage notes:

MapTimeCDefault controls the C type that is used when SQL_C_DEFAULT is specified for TIME columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which assume SQL_C_CHAR as the default C type for datetime values. Set MapTimeCDefault as follows:

- 0 - for the default SQL_C_TYPE_TIME C type representation: a struct containing numeric members for hour, minute, and second
- 1 - for an SQL_C_CHAR C type representation: "12:34:56"
- 2 - for an SQL_C_WCHAR C type representation: "12:34:56" in UTF-16.

This keyword affects the behavior of CLI functions where SQL_C_DEFAULT may be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

Note: MapTimeCDefault supersedes Patch2=24. If both MapTimeCDefault and Patch2=24 are set, the MapTimeCDefault value takes precedence.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “CLI and ODBC function summary” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “MapDateCDefault CLI/ODBC configuration keyword” on page 336
- “MapTimestampCDefault CLI/ODBC configuration keyword” on page 342
- “Patch2 CLI/ODBC configuration keyword” on page 352

MapTimeDescribe CLI/ODBC configuration keyword

Keyword description:

Controls the SQL data type returned when TIME columns and parameter markers are described.

db2cli.ini keyword syntax:

MapTimeDescribe = 0 | 1 | 2

Default setting:

The default SQL data type for TIME data is returned: SQL_TIME for ODBC 2.0 or SQL_TYPE_TIME for ODBC 3.0

Usage notes:

To control the SQL data type that is returned when TIME columns and parameter markers are described, set MapTimeDescribe as follows:

- 0 - to return the default SQL data type: SQL_TIME for ODBC 2.0 or SQL_TYPE_TIME for ODBC 3.0
- 1 - to return the SQL_CHAR SQL data type
- 2 - to return the SQL_WCHAR SQL data type

Only the following DB2 CLI functions are affected by setting MapTimeDescribe:

- SQLColumns()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()
- SQLProcedureColumns()
- SQLSpecialColumns()

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “SQLColumns function (CLI) - Get column information for a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeParam function (CLI) - Return description of a parameter marker” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSpecialColumns function (CLI) - Get special (row identifier) columns” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “MapDateDescribe CLI/ODBC configuration keyword” on page 337
- “MapGraphicDescribe CLI/ODBC configuration keyword” on page 339
- “MapTimestampDescribe CLI/ODBC configuration keyword” on page 342

MapTimestampCDefault CLI/ODBC configuration keyword

Keyword description:

Specify the default C type of TIMESTAMP columns and parameter markers.

db2cli.ini keyword syntax:

MapTimestampCDefault = 0 | 1 | 2

Default setting:

The default C type representation for TIMESTAMP data is SQL_C_TYPE_TIMESTAMP.

Usage notes:

MapTimestampCDefault controls the C type that is used when SQL_C_DEFAULT is specified for TIMESTAMP columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which assume SQL_C_CHAR as the default C type for datetime values. Set MapTimestampCDefault as follows:

- 0 - for the default SQL_C_TYPE_TIMESTAMP C type representation: a struct containing numeric members for year, month, day, hour, minute, second, and fraction of a second
- 1 - for an SQL_C_CHAR C type representation: "2004-01-01 12:34:56.123456"
- 2 - for an SQL_C_WCHAR C type representation: "2004-01-01 12:34:56.123456" in UTF-16.

This keyword affects the behavior of CLI functions where SQL_C_DEFAULT may be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

Related concepts:

- "db2cli.ini initialization file" on page 277
- "Data types and data conversion in CLI applications" on page 39

Related reference:

- "CLI and ODBC function summary" in *Call Level Interface Guide and Reference, Volume 2*
- "CLI/ODBC configuration keywords listing by category" on page 279
- "MapDateCDefault CLI/ODBC configuration keyword" on page 336
- "MapTimeCDefault CLI/ODBC configuration keyword" on page 340

MapTimestampDescribe CLI/ODBC configuration keyword

Keyword description:

Controls the SQL data type returned when TIMESTAMP columns and parameter markers are described.

db2cli.ini keyword syntax:

MapTimestampDescribe = 0 | 1 | 2

Default setting:

The default SQL data type for TIMESTAMP data is returned: SQL_TIMESTAMP for ODBC 2.0 or SQL_TYPE_TIMESTAMP for ODBC 3.0.

Usage notes:

To control the SQL data type that is returned when `TIMESTAMP` columns and parameter markers are described, set `MapTimestampDescribe` as follows:

- 0 - to return the default SQL data type: `SQL_TIMESTAMP` for ODBC 2.0 or `SQL_TYPE_TIMESTAMP` for ODBC 3.0
- 1 - to return the `SQL_CHAR` SQL data type
- 2 - to return the `SQL_WCHAR` SQL data type

Only the following DB2 CLI functions are affected by setting `MapTimeStampDescribe`:

- `SQLColumns()`
- `SQLDescribeCol()`
- `SQLDescribeParam()`
- `SQLGetDescField()`
- `SQLGetDescRec()`
- `SQLProcedureColumns()`
- `SQLSpecialColumns()`

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Data types and data conversion in CLI applications” on page 39

Related reference:

- “SQLColumns function (CLI) - Get column information for a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeCol function (CLI) - Return a set of attributes for a column” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLDescribeParam function (CLI) - Return description of a parameter marker” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescField function (CLI) - Get single field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLSpecialColumns function (CLI) - Get special (row identifier) columns” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “MapDateDescribe CLI/ODBC configuration keyword” on page 337
- “MapGraphicDescribe CLI/ODBC configuration keyword” on page 339
- “MapTimeDescribe CLI/ODBC configuration keyword” on page 340

MapXMLCDefault CLI/ODBC configuration keyword

Keyword description:

Controls the default C type representation used when `SQL_C_DEFAULT` is specified for XML columns and parameter markers.

db2cli.ini keyword syntax:

MapXMLCDefault = 0 | 1 | 2 | 3

Default setting:

The default C type representation for XML data is SQL_C_BINARY.

Usage notes:

MapXMLCDefault controls the C type that is used when SQL_C_DEFAULT is specified for XML columns and parameter markers. This keyword should be used primarily with Microsoft applications, such as Microsoft Access, which might assume SQL_C_WCHAR as the default C type for XML values. Set MapXMLCDefault as follows:

- 0 - for the default SQL_C_BINARY C type representation
- 1 - for the SQL_C_CHAR C type representation; this can result in data loss as the XML data is converted to the local application code page
- 2 - for the SQL_C_WCHAR C type representation

This keyword affects the behaviour of CLI functions where SQL_C_DEFAULT can be specified as a C type, such as SQLBindParameter(), SQLBindCol(), and SQLGetData().

Related concepts:

- “db2cli.ini initialization file” on page 277
- “XML data handling in CLI applications - Overview” on page 117
- “XML data type” in *XML Guide*

Related reference:

- “C data types for CLI applications” on page 43
- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*

MapXMLDescribe CLI/ODBC configuration keyword

Keyword description:

Controls the SQL data type returned when XML columns and parameter markers are described.

db2cli.ini keyword syntax:

MapXMLDescribe = -370 | -350 | -152 | -99 | -98

Default setting:

The default SQL data type for XML data is returned: SQL_XML (-370)

Usage notes:

To control the SQL data type that is returned when XML columns and parameter markers are described, set MapXMLDescribe to one of the following integer values:

- -370 to return the default SQL_XML SQL data type

- -350 to return the SQL_DBCLOB SQL data type
- -152 to return the SQL_SS_XML SQL data type

Note: The SQL_SS_XML value of -152 belongs to the reserved range of Microsoft SQL Server and is not defined by IBM.

- -99 to return the SQL_BLOB SQL data type
- -98 to return the SQL_CLOB SQL data type

The data length for XML values mapped to LOB types is the maximum length for the mapped data type.

When used in conjunction with the LongDataCompat keyword set to the value 1, XML values mapped to LOB data types will be mapped to the corresponding LONG data type as well.

Character types specified for MapXMLDescribe may result in data loss during data conversion if the application code page does not support all of the characters in the source data. Mapping XML values to character types, therefore, is only recommended with caution.

This keyword is recommended to provide compatibility with applications that access XML columns as CLOB or BLOB, or use Microsoft application development technologies.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “XML data encoding” in *XML Guide*
- “XML data handling in CLI applications - Overview” on page 117
- “XML data type” in *XML Guide*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “LongDataCompat CLI/ODBC configuration keyword” on page 333
- “SQL symbolic and default data types for CLI applications” on page 41

MaxLOBBlockSize CLI/ODBC configuration keyword

Keyword description:

Maximum size of LOB or XML data block.

db2cli.ini keyword syntax:

MaxLOBBlockSize = 0 | ... | 2147483647

Default setting:

There is no limit to the data block size for LOB or XML data.

Equivalent connection or statement attribute:

SQL_ATTR_MAX_LOB_BLOCK_SIZE

Usage notes:

Specify the maximum return block size for LOB or XML data.

During data retrieval, the server will include all of the information for the current row in its reply to the client even if the maximum block size has been reached.

If both MaxLOBBlockSize and the db2set registry variable DB2_MAX_LOB_BLOCK_SIZE are specified, the value for MaxLOBBlockSize will be used.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Miscellaneous variables” in *Performance Guide*

Mode CLI/ODBC configuration keyword

Keyword description:

Default connection mode.

db2cli.ini keyword syntax:

Mode = SHARE | EXCLUSIVE

Default setting:

SHARE

Not applicable when:

connecting to a host or iSeries server.

Usage notes:

Sets the CONNECT mode to either SHARE or EXCLUSIVE. If a mode is set by the application at connect time, this value is ignored. The default is SHARE.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “CONNECT (Type 1) statement” in *SQL Reference, Volume 2*

NotifyLevel CLI/ODBC configuration keyword

Keyword description:

Indicates the diagnostic level.

db2cli.ini keyword syntax:

NotifyLevel = 0 | 1 | 2 | 3 | 4

Default setting:

3

Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is equivalent to the database manager parameter NOTIFYLEVEL. See notifylevel - Notify level configuration parameter for a details about the values.

OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword

Keyword description:

Makes OLE DB flag LONG data types with DBCOLUMNFLAGS_ISLONG.

db2cli.ini keyword syntax:

OleDbReportIsLongForLongTypes = 0 | 1

Equivalent connection attribute:

SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB

Default setting:

LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) do not have the DBCOLUMNFLAGS_ISLONG flag set, which might cause the columns to be used in the WHERE clause.

Usage notes:

The OLE DB client cursor engine and the OLE DB .NET Data Provider CommandBuilder object generate UPDATE and DELETE statements based on column information provided by the IBM DB2 OLE DB Provider. If the generated statement contains a LONG type in the WHERE clause, the statement will fail because LONG types cannot be used in a search with an equality operator. Setting the keyword OleDbReportIsLongForLongTypes to 1 will make the IBM DB2 OLE DB Provider report LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) with the DBCOLUMNFLAGS_ISLONG flag set. This will prevent the long columns from being used in the WHERE clause.

The OleDbReportIsLongForLongTypes keyword is supported by the following database servers:

- DB2 for z/OS
 - version 6 with PTF UQ93891
 - version 7 with PTF UQ93889
 - version 8 with PTF UQ93890
 - versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
 - version 8.2 (equivalent to Version 8.1, FixPak 7) and later

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Data type mappings between DB2 and OLE DB” in *Developing ADO.NET and OLE DB Applications*
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

OleDbReturnCharAsWChar CLI/ODBC configuration keyword

Keyword description:

Controls how the IBM DB2 OLE DB Provider describes CHAR, VARCHAR, LONG VARCHAR, and CLOB data.

db2cli.ini keyword syntax:

OleDbReturnCharAsWChar = 0 | 1

Default setting:

The IBM DB2 OLE DB Provider describes CHAR, VARCHAR, LONG VARCHAR, and CLOB data as DBTYPE_WSTR.

Usage notes:

The IBM DB2 OLE DB Provider describes CHAR, VARCHAR, LONG VARCHAR, and CLOB data as DBTYPE_WSTR by default as of DB2 UDB Version 8.1.2. The CLI/ODBC configuration keyword OleDbReturnCharAsWChar allows you to change this default to have the previously stated character data types reported as DBTYPE_STR.

The available settings are:

- 0 - CHAR, VARCHAR, LONG VARCHAR, and CLOB data are described as DBTYPE_STR, and the code page of data in ISequentialStream is the local code page of the client
- 1 - CHAR, VARCHAR, LONG VARCHAR, and CLOB data are reported as DBTYPE_WSTR, and the code page of data in ISequentialStream is UCS-2

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Purpose of the IBM OLE DB Provider for DB2” in *Developing ADO.NET and OLE DB Applications*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Data type mappings between DB2 and OLE DB” in *Developing ADO.NET and OLE DB Applications*

OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword

Keyword description:

Makes OLE DB's IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) return a row set sorted by the ORDINAL_POSITION column.

db2cli.ini keyword syntax:

OleDbSQLColumnsSortByOrdinal = 0 | 1

Equivalent connection attribute:

SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB

Default setting:

IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) returns the row set sorted by the columns TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME.

Usage notes:

The Microsoft OLE DB specification requires that

IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) returns the row set sorted by the columns TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME. The IBM DB2 OLE DB Provider conforms to the specification. However, applications that use the Microsoft ODBC Bridge provider (MSDASQL) have been typically coded to get the row set sorted by ORDINAL_POSITION. Setting the OleDbSQLColumnsSortByOrdinal keyword to 1 will make the provider return a row set sorted by ORDINAL_POSITION.

The OleDbSQLColumnsSortByOrdinal keyword is supported by the following database servers:

- DB2 for z/OS
 - version 6 with PTF UQ93891
 - version 7 with PTF UQ93889
 - version 8 with PTF UQ93890
 - versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
 - version 8.2 (equivalent to Version 8.1, FixPak 7) and later

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Schema rowsets supported by the IBM OLE DB Provider” in *Developing ADO.NET and OLE DB Applications*
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

OptimizeForNRows CLI/ODBC configuration keyword

Keyword description:

Append 'OPTIMIZE FOR n ROWS' clause to every select statement.

db2cli.ini keyword syntax:

OptimizeForNRows = *integer*

Default setting:

The clause is not appended.

Equivalent statement attribute:

SQL_ATTR_OPTIMIZE_FOR_NROWS

Usage notes:

This option will append the "OPTIMIZE FOR n ROWS" clause to every select statement, where n is an integer larger than 0. If set to 0 (the default) this clause will not be appended.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SELECT statement” in *SQL Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Patch1 CLI/ODBC configuration keyword

Keyword description:

Use work-arounds for known CLI/ODBC application problems.

db2cli.ini keyword syntax:

Patch1 = { 0 | 1 | 2 | 4 | 8 | 16 | ... }

Default setting:

Use no work-arounds.

Usage notes:

This keyword is used to specify a work-around for known problems with ODBC applications. The value specified can be for none, one, or multiple work-arounds. The patch values specified here are used in conjunction with any Patch2 values that might also be set.

Using the DB2 CLI/ODBC Settings notebook you can select one or more patches to use. If you set the values in the `db2cli.ini` file itself and want to use multiple patch values then simply add the values together to form the keyword value. For example, if you want the patches 1, 4, and 8, then specify `Patch1=13`.

- 0 = No work around (default)

Table 25. Patch1 CLI/ODBC configuration keyword values

Value	Description
4	Changes input timestamp data to date data if the time and fraction part of the timestamp are zero. For example, {ts 'YYYY-MM-DD 00:00:00'} is changed to {d 'YYYY-MM-DD'}. This value is typically needed for older versions of Microsoft Access.
8	Changes input timestamp data to time data if the date part of the timestamp is either 1899-12-30 or 1900-01-01. For example, {ts '1899-12-30 HH:MM:SS'} is changed to {t 'HH:MM:SS'}. This value is typically needed for older versions of Microsoft Access.
64	Null-terminates output GRAPHIC strings. This value is typically needed by Microsoft Access in a double-byte (DBCS) environment.

Table 25. Patch1 CLI/ODBC configuration keyword values (continued)

Value	Description
128	<p>Disables the default performance optimization behavior for the MSysConf table associated with some Microsoft applications.</p> <p>Microsoft applications, such as Microsoft Access, use a configuration table called MSysConf. Once these applications successfully connect to a database, they will typically issue the following query: "SELECT Config, nValue FROM MSysConf". Because the MSysConf table does not exist in a DB2 database by default, this query fails with the error "SQL0204N "MSysConf" is an undefined name.". Microsoft applications can handle this error and continue processing, however, issuing the query across the network to the DB2 server incurs overhead.</p> <p>To enhance performance, DB2 CLI assumes that this query will always fail, so when it detects that an application is trying to execute this query, it automatically returns an error with an SQLSTATE of S0002 (Table not found). The query, therefore, is never sent to the server. If, however, the user has created the MSysConf configuration table in the database and wants the application to access it, this PATCH1 value can be set to disable the performance optimization and allow the query to be executed.</p>
256	Service use only
512	Service use only
1024	Returns SQL_SUCCESS_WITH_INFO instead of SQL_NO_DATA_FOUND from the SQLExecute() and SQLExecDirect() functions if the executed UPDATE or DELETE statement affected no rows. This value might be needed by some Microsoft Visual Basic applications.
4096	Prevents a COMMIT from being issued after closing a cursor in autocommit mode.
8192	Returns an extra result set after invoking a stored procedure. This extra result set has one row and consists of the output values of the stored procedure. This PATCH1 value might be needed by some Powerbuilder applications that require an extra result set.
32768	Forces the driver to make Microsoft Query applications work with DB2 MVS synonyms.
65536	Deprecated
131072	Deprecated
262144	Deprecated

Table 25. Patch1 CLI/ODBC configuration keyword values (continued)

Value	Description
524288	Deprecated
1048576	Service use only
2097152	Service use only

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Patch2 CLI/ODBC configuration keyword” on page 352

Patch2 CLI/ODBC configuration keyword

Keyword description:

Use work-arounds for known CLI/ODBC application problems.

db2cli.ini keyword syntax:

Patch2 = "patch value 1, patch value 2, patch value 3, ..."

Default setting:

Use no work-arounds

Usage notes:

This keyword is used to specify a work-around for known problems with CLI/ODBC applications. The value specified can be for none, one, or multiple work-arounds. The patch values specified here may be used in conjunction with any Patch1 values that may also be set.

When specifying multiple patches, the values are specified in a comma delimited string (unlike the Patch1 option where the values are added together and the sum is used).

- 0 = No work around (default)

To set Patch2 values 3, 4 and 8 you would specify:

Patch2="3, 4, 8"

Table 26. Patch2 CLI/ODBC configuration keyword values

Value	Description
1	Deprecated
3	Service use only
4	Deprecated
5	Deprecated
6	Forces the driver to return a message indicating that scrollable cursors are not supported. This setting is needed by some applications (such as Visual Basic) that make use of LOBs or that do not need or want scrollable cursors to be used, even though they have been explicitly requested by the application.

Table 26. Patch2 CLI/ODBC configuration keyword values (continued)

Value	Description
7	Maps all GRAPHIC column data types to the CHAR column data type. The precision of a GRAPHIC column will also be doubled; for example, GRAPHIC(20) will be reported as CHAR(40).
8	Ignores catalog search arguments in schema calls.
11	SQLGetInfo() reports that catalog names are supported for Visual Basic stored procedures.
12	Deprecated
13	Prevents keywords in the db2cli.ini initialization file from being appended to the output connection string.
14	Deprecated
15	Causes a period separator to be used instead of the default locale's decimal separator in character output.
16	Deprecated
17	Deprecated
18	Attempts to replace literals with parameter markers for inefficient applications that use literals repeatedly. It is only applicable to INSERT SQL statements with the VALUES clause using only literals. Coding your application properly to use parameter markers is the best solution.
19	Removes parentheses from the ON clause of an outer join, where the outer join is an ODBC escape sequence and the server is DB2 for MVS Version 5. DB2 for MVS Version 5 does not currently support the ODBC syntax where parentheses are permitted in the ON clause of an outer join clause. Setting this PATCH2 value allows the outer join escape sequence to be used against DB2 for MVS Version 5. This value should only be set when the server is DB2 for MVS Version 5.
20	Forces the driver to rewrite the BETWEEN predicate when the server is DB2 for MVS. DB2 for MVS does not currently support the BETWEEN predicate with parameter markers as both operands. Setting this PATCH2 value causes (expression ? BETWEEN ?) to be rewritten as (expression >= ? and expression <= ?).
21	Deprecated

Table 26. Patch2 CLI/ODBC configuration keyword values (continued)

Value	Description
22	Causes SQLGetInfo() to report SQL_OUTER_JOINS=NO and SQL_OJ_CAPABILITIES=0. This prevents the application from using outer joins where they are not supported, thus ensuring that the outer join queries do not fail.
23	Deprecated
24	Reports TIME data as SQL_CHAR data. This patch value is used as a workaround for Microsoft Access applications.
25	Removes trailing zeros in the CHAR representation of DECIMAL columns; used as a workaround for Microsoft Access applications.
28	Deprecated
29	Removes leading zeroes in the string representation of DECIMAL values x, where $1 > x > -1$; used as a workaround for ADO applications with some MDAC versions.
30	Disables stored procedure caching optimization.
31	Deprecated
32	Deprecated
33	Returns the ISO version of timestamp data when converted to CHAR, rather than the ODBC version.
34	Deprecated
38	Turns statement caching off
42	Prevents the FOR UPDATE clause from being used with keyset cursors. By default, most applications expect keyset cursors to be updatable, however, if this is not required, then this PATCH2 value makes the cursor read-only (but still scrollable and sensitive to changes made by others).
50	Frees LOB locators when SQLFetch() is executed, rather than when a COMMIT is issued. This PATCH2 value frees the locators that are used internally when applications fetch LOB data without binding the LOB columns with SQLBindCol() (or equivalent descriptor APIs). Locators that are explicitly returned to the application must still be freed by the application. This PATCH2 value can be used to avoid scenarios where an application receives SQLCODE = -429 (no more locators).
56	Allows client support for Early Close Cursors for those servers that do not support it as in the case of DB2 UDB for OS/390 version 7 or earlier.

Table 26. Patch2 CLI/ODBC configuration keyword values (continued)

Value	Description
57	Allows calling a stored procedure that returns a NULL output parameter value without providing an output indicator pointer. This is normally applicable to older versions of Borland Delphi products.
58	Date/Time values inserted into the database that cause truncation errors can be downgraded to a truncation warning using this PATCH2 value.
61	When data is given to the client from an SQL_CHAR there may be right padded spaces. This patch value strips off right padded single byte spaces, but not double byte spaces. This behavior partially mimics the Neon Shadow Driver behavior

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Patch1 CLI/ODBC configuration keyword” on page 350

Port CLI/ODBC configuration keyword

Keyword description:

The server system’s service name or port number, used with a file DSN or in a DSN-less connection.

db2cli.ini keyword syntax:

Port = *service name* | *port number*

Default setting:

None

Only applicable when:

Protocol set to TCPIP

Usage notes:

Use this option in conjunction with the Hostname option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the Protocol option is set to TCPIP.

Specify either the server system’s service name or its port number. The service name must be available for lookup at the client machine if it is used.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Hostname CLI/ODBC configuration keyword” on page 327
- “Protocol CLI/ODBC configuration keyword” on page 356
- “ServiceName CLI/ODBC configuration keyword” on page 366

ProgramName CLI/ODBC configuration keyword

Keyword description:

Change the default client application name to a user-defined name which is used to identify the application at the server when monitoring.

db2cli.ini keyword syntax:

ProgramName = <string> | PID

Default setting:

No user-defined name is used. The application is identified by the name DB2 assigns by default.

Equivalent connection attribute:

SQL_ATTR_INFO_PROGRAMNAME

Usage notes:

When monitoring a CLI application, it may be useful to identify the application by a user-defined string, instead of by the default identifier that DB2 assigns. ProgramName allows the user to specify the identifier as either a string up to 80 bytes in length or the string "PID" (without the quotation marks).

If ProgramName is set to "PID" for a CLI application, the application's name will consist of the prefix "CLI" along with the application's process ID and the current active connection handle, as follows: CLI<pid>:<connectionHandle#>. The "PID" setting is useful when monitoring application servers that run multiple applications with numerous connections to the same database.

(When the ProgramName keyword is set to "PID" for other types of applications, the "CLI" prefix is replaced with the following values corresponding to the type of application: "JDBC" for JDBC applications, "OLEDB" for OLE DB applications, and "ADONET" for .NET applications.)

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Protocol CLI/ODBC configuration keyword

Keyword description:

Communications protocol used for File DSN or in a DSN-less connection.

db2cli.ini keyword syntax:

Protocol = TCP/IP | TCP/IP6 | TCP/IP4 | IPC | LOCAL

Default setting:

none

Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

TCP/IP is the only protocol supported when using a File DSN. Set the option to the string `TCPIP` (without the slash).

When this option is set then the following options must also be set:

- Database;
- ServiceName; and
- Hostname.

IPC connectivity can be specified by setting Protocol to either **IPC** or **LOCAL**.

When Protocol = **IPC** | **LOCAL** the Instance keyword must also be set.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Database CLI/ODBC configuration keyword” on page 309
- “Hostname CLI/ODBC configuration keyword” on page 327
- “ServiceName CLI/ODBC configuration keyword” on page 366

PWD CLI/ODBC configuration keyword

Keyword description:

Define default password.

db2cli.ini keyword syntax:

`PWD = password`

Default setting:

None

Usage notes:

This *password* value is used if a password is not provided by the application at connect time.

It is stored as plain text in the `db2cli.ini` file and is therefore not secure.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “UID CLI/ODBC configuration keyword” on page 388

PWDPlugin CLI/ODBC configuration keyword

Keyword description:

Password plugin to be used for client side authentication for file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

PWDPlugin = *plugin name*

Default setting:

By default, the value is null and the DB2-supplied userid-password plug-in library is used.

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies the name of the userid-password plug-in library to be used for client-side connection authentication. The plug-in is used when the client is authenticated using SERVER or SERVER_ENCRYPT authentication.

QueryTimeoutInterval CLI/ODBC configuration keyword

Keyword Description:

Delay (in seconds) between checks for a query timeout.

db2cli.ini Keyword Syntax:

QueryTimeoutInterval = 0 | 5 | **positive integer**

Default Setting:

5 seconds

Usage Notes:

An application can use the SQLSetStmtAttr() function to set the SQL_ATTR_QUERY_TIMEOUT statement attribute. This attribute indicates the number of seconds to wait for an SQL statement or XQuery expression to complete executing before attempting to cancel the execution and returning to the application.

The QueryTimeoutInterval configuration keyword is used to indicate how long the CLI driver should wait between checks to see if the query has completed.

For instance, suppose SQL_ATTR_QUERY_TIMEOUT is set to 25 seconds (timeout after waiting for 25 seconds), and QueryTimeoutInterval is set to 10 seconds (check the query every 10 seconds). The query will not time out until 30 seconds (the first check AFTER the 25 second limit).

Note: DB2 CLI implements query timeout by starting a thread that periodically queries the status of each executing query. The QueryTimeoutInterval value specifies how long the query timeout thread waits between checks for expired queries. Because this is an asynchronous operation to the queries being executed, it is possible that a given query may not be timed out until SQL_ATTR_QUERY_TIMEOUT + QueryTimeoutInterval seconds. In the example above, the best-case timeout would be at 26 seconds, and the worst-case timeout would be at 35 seconds.

There may be cases where the `SQL_ATTR_QUERY_TIMEOUT` is set to a value which is too low, and the query should NOT be timed-out. If the application cannot be modified (that is, a third party ODBC application), then the `QueryTimeoutInterval` can be set to 0, and the CLI driver will ignore the `SQL_ATTR_QUERY_TIMEOUT` setting, and therefore wait for SQL statements to complete execution before returning to the application.

Note: If `QueryTimeoutInterval` is set to 0, any attempt by the application to set `SQL_ATTR_QUERY_TIMEOUT` will result in `SQLSTATE 01S02` (Option Value Changed).

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Alternatively, `QueryTimeoutInterval` can be set to a value that is larger than the `SQL_ATTR_QUERY_TIMEOUT` setting, thus preventing timeouts from occurring at the specified interval. For example, if the application sets a 15 second `SQL_ATTR_QUERY_TIMEOUT` value, but the server requires at least 30 seconds to execute the query, the `QueryTimeoutInterval` can be set to a value of 30 seconds or so to prevent this query from timing out after 15 seconds.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLSetStmtAttr function (CLI) - Set options related to a statement” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword

Keyword description:

Allow a NULL connect to process the COMMON section of the db2cli.ini initialization file.

db2cli.ini keyword syntax:

`ReadCommonSectionOnNullConnect = 0 | 1`

Default setting:

A NULL connect does not process the db2cli.ini initialization file.

Usage notes:

For use with DB2 CLI, DB2 JDBC Type 2 and DB2 .NET stored procedures, specify 1 to allow stored procedures to read the COMMON section of the db2cli.ini file, thus allowing stored procedures to use keywords listed in that section.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

Related concepts:

- “Routines: Procedures” in *SQL Guide*
- “db2cli.ini initialization file” on page 277

Related tasks:

- “Creating routines” in *SQL Guide*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

ReceiveTimeout CLI/ODBC configuration keyword

Keyword description:

Specify the time in seconds to wait for a reply from the server on an established connection before terminating the attempt and generating a communication timeout error.

db2cli.ini keyword syntax:

ReceiveTimeout = 0 | 1 | 2 | ... | 32767

Default setting:

The client waits indefinitely for a reply from the server on an established connection.

Equivalent connection attribute:

SQL_ATTR_RECEIVE_TIMEOUT

Usage notes:

This keyword has no effect during connection establishment and is only supported for TCP/IP protocol.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

Reopt CLI/ODBC configuration keyword

Keyword description:

Enable query optimization or reoptimization of SQL statements that have special registers or parameter markers.

db2cli.ini keyword syntax:

Reopt = 2 | 3 | 4

Default setting:

No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for special registers or parameter markers.

Equivalent connection or statement attribute:

SQL_ATTR_REOPT

Usage notes:

Optimization occurs by using the values available at query execution time for the special registers or parameter markers instead of the default estimates that are chosen by the compiler. The valid values of the keyword are:

- 2 = SQL_REOPT_NONE. This is the default. No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for the special registers or parameter markers. The default NULLID package set is used to execute dynamic SQL statements.
- 3 = SQL_REOPT_ONCE. Query optimization occurs once at query execution time, when the query is executed for the first time. The NULLIDR1 package set, which is bound with the REOPT ONCE bind option, is used.
- 4 = SQL_REOPT_ALWAYS. Query optimization or reoptimization occurs at query execution time every time the query is executed. The NULLIDRA package set, which is bound with the REOPT ALWAYS bind option, is used.

The NULLIDR1 and NULLIDRA are reserved package set names, and when used, REOPT ONCE and REOPT ALWAYS are implied respectively. These package sets have to be explicitly created with the following commands:

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

If both the Reopt and CurrentPackageSet keywords are specified, CurrentPackageSet takes precedence.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “CLI/ODBC configuration keywords listing by category” on page 279
- “CurrentPackageSet CLI/ODBC configuration keyword” on page 305
- “BIND command” in *Command Reference*

ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword

Keyword description:

Return errors that were uncovered during DB2 CLI error recovery as warnings.

db2cli.ini keyword syntax:

ReportRetryErrorsAsWarnings = 0 | 1

Only applicable when:

RetryOnError keyword is set to 1.

Default setting:

Do not return errors uncovered during DB2 CLI error recovery to the application.

Usage notes:

By default, when the DB2 CLI retry logic is able to recover successfully from a non-fatal error, it masks that error from the application by returning

SQL_SUCCESS. Because application binding errors can be hidden this way, for debugging purposes, you may want to set ReportRetryErrorsAsWarnings to 1. This setting keeps the error recovery on, but forces DB2 CLI to return to the application, any errors that were uncovered as warnings.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI function return codes” on page 50
- “CLI/ODBC configuration keywords listing by category” on page 279
- “RetryOnError CLI/ODBC configuration keyword” on page 364

ReportPublicPrivileges CLI/ODBC configuration keyword

Keyword description:

Report PUBLIC privileges in SQLColumnPrivileges() and SQLTablePrivileges() results.

db2cli.ini keyword syntax:

ReportPublicPrivileges = 0 | 1

Default setting:

PUBLIC privileges are not reported.

Usage notes:

This keyword specifies if privileges assigned to the PUBLIC group are to be reported as if PUBLIC was a user in the SQLColumnPrivileges() and SQLTablePrivileges() results.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLTablePrivileges function (CLI) - Get privileges associated with a table” in *Call Level Interface Guide and Reference, Volume 2*

RetCatalogAsCurrServer CLI/ODBC configuration keyword

Keyword description:

Catalog functions return the CURRENT SERVER value instead of the null value for the catalog columns.

db2cli.ini keyword syntax:

RetCatalogAsCurrServer= 0 | 1

Default setting:

If the target DBMS returns null for the catalog columns, the CURRENT SERVER value will not be substituted.

Usage notes:

If the catalog functions for the target DBMS return a null value for the

catalog columns, setting `RetCatalogAsCurrServer` to 1 causes the DBMS to return the `CURRENT SERVER` value instead.

- 0 = Catalog functions return the null value for the catalog columns (default).
- 1 = Catalog functions return the `CURRENT SERVER` value, instead of the null value, for the catalog columns.

For example, assume the catalog function `SQLTables()` returns a result set where the values in the `TABLE_CAT` column are null values. Setting `RetCatalogAsCurrServer` to 1 causes the DBMS to return the `CURRENT SERVER` value in the `TABLE_CAT` column.

Related concepts:

- “`db2cli.ini` initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “`SQLTables` function (CLI) - Get table information” in *Call Level Interface Guide and Reference, Volume 2*

RetOleDbConnStr CLI/ODBC configuration keyword

Keyword description:

Specify whether the Mode CLI/ODBC configuration keyword returns a numeric or string value.

db2cli.ini keyword syntax:

`RetOleDbConnStr = 0 | 1`

Default setting:

The value for the Mode CLI/ODBC configuration keyword is returned as a string.

Usage notes:

The Mode CLI/ODBC configuration keyword sets the `CONNECT` mode to either `SHARE` or `EXCLUSIVE`. OLE DB expects the value for Mode to have a numeric representation instead of a string representation.

`RetOleDbConnStr` toggles between returning a string and a numeric value.

The possible settings are as follows:

- 0 — the value returned by `SQLDriverConnect()` and `SQLBrowseConnect()` for the Mode keyword is either `SHARE` or `EXCLUSIVE`
- 1 — the value returned by `SQLDriverConnect()` and `SQLBrowseConnect()` for the Mode keyword is either 3 (for `SHARE`) or 12 (for `EXCLUSIVE`)

For example, if you set `RetOleDbConnStr=1` and call `SQLDriverConnect()` or `SQLBrowseConnect()` with the following input connection string for a shared connection:

```
DSN=SAMPLE;MODE=SHARE
```

then the output connection string will have the following format:

```
DSN=SAMPLE;UID=;PWD=;MODE=3
```

If you set `RetOleDbConnStr=1` and call `SQLDriverConnect()` or `SQLBrowseConnect()` with the following input connection string for an exclusive connection:

```
DSN=SAMPLE;UID=NEWTON;PWD=SECRET;MODE=EXCLUSIVE
```

then the output connection string will have the following format:

```
DSN=SAMPLE;UID=NEWTON;PWD=SECRET;MODE=12
```

OLE DB applications that use the string representation for the value of the `Mode` keyword returned by `SQLDriverConnect()` and `SQLBrowseConnect()` will receive an error from OLE DB Component Services. OLE DB Component Services returns an error because it expects the keyword `Mode` to have numeric values. Setting `RetOleDbConnStr` to 1 avoids this behavior, as the value for `Mode` will then be numeric.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Mode CLI/ODBC configuration keyword” on page 346

RetryOnError CLI/ODBC configuration keyword

Keyword description:

Turn on or off the DB2 CLI driver’s error recovery behavior.

db2cli.ini keyword syntax:

```
RetryOnError = 0 | 1
```

Default setting:

Allow the DB2 CLI driver to attempt error recovery on non-fatal errors.

Usage notes:

By default, DB2 CLI will attempt to recover from non-fatal errors, such as incorrect binding of application parameters, by retrieving additional information on the failing SQL statement and then executing the statement again. The additional information retrieved includes input parameter information from the database catalog tables. If DB2 CLI is able to recover successfully from the error, by default, it does not report the error to the application. The CLI/ODBC configuration keyword `ReportRetryErrorsAsWarnings` allows you to set whether error recovery warnings are returned to the application or not.

Important: Once DB2 CLI has successfully completed the error recovery, the application may behave differently, because DB2 CLI will use the catalog information gathered during the recovery for subsequent executions of that particular SQL statement, rather than the information provided in the original `SQLBindParameter()` function calls. If you do not want this behavior, set `RetryOnError` to 0, forcing DB2 CLI not to attempt recovery. You should, however, modify the application to correctly bind statement parameters.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related tasks:

- “Binding parameter markers in CLI applications” on page 29

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword” on page 361
- “SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator” in *Call Level Interface Guide and Reference, Volume 2*

SaveFile CLI/ODBC configuration keyword

Keyword description:

The file name of a DSN file in which to save the attribute values of the keywords used in making the present, successful connection.

db2cli.ini keyword syntax:

You can not set this keyword in the `db2cli.ini` file.

You can specify the value of this keyword in the connection string in `SQLDriverConnect` like this:

`SaveFile = file name`

SchemaList CLI/ODBC configuration keyword

Keyword description:

Restrict schemas used to query table information.

db2cli.ini keyword syntax:

`SchemaList = " 'schema1', 'schema2',... 'schemaN' "`

Default setting:

None

Usage notes:

`SchemaList` is used to provide a more restrictive default, and therefore improve performance, for those applications that list every table in the DBMS.

If there are a large number of tables defined in the database, a schema list can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application. Each schema name is case-sensitive, must be delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
SchemaList="'USER1','USER2','USER3'"
```

For DB2 for z/OS, `CURRENT SQLID` can also be included in this list, but without the single quotes, for example:

```
SchemaList="'USER1',CURRENT SQLID,'USER3'"
```

The maximum length of the string is 256 characters.

This option can be used in conjunction with `DBName` and `TableType` to further limit the number of tables for which information will be returned.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “DBName CLI/ODBC configuration keyword” on page 314
- “TableType CLI/ODBC configuration keyword” on page 374

ServerMsgMask CLI/ODBC configuration keyword

Keyword description:

Indicate when DB2 CLI should request the error message from the server.

db2cli.ini keyword syntax:

ServerMsgMask = 0 | 1 | -2 | -1

Default setting:

DB2 CLI will check the local message files first to see if the message can be retrieved. If no matching SQLCODE is found, DB2 CLI will request the information from the server.

Equivalent connection attribute:

SQL_ATTR_SERVER_MSGTXT_MASK

Usage notes:

This keyword is used in conjunction with the UseServerMsgSP CLI/ODBC configuration keyword. The keyword can be set to:

- **0 (default)** = SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST. DB2 CLI will check the local message files first to see if the message can be retrieved. If no matching SQLCODE is found, DB2 CLI will request the information from the server.
- **1** = SQL_ATTR_SERVER_MSGTXT_MASK_WARNINGS. DB2 CLI always requests the message information from the server for warnings but error messages are retrieved from the local message files.
- **-2** = SQL_ATTR_SERVER_MSGTXT_MASK_ERRORS. DB2 CLI always requests the message information from the server for errors but warning messages are retrieved from the local message files.
- **-1** = SQL_ATTR_SERVER_MSGTXT_MASK_ALL. DB2 CLI always requests the message information from the server for both error and warning messages.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “UseServerMsgSP CLI/ODBC configuration keyword” on page 391
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

ServiceName CLI/ODBC configuration keyword

Keyword description:

The server system’s service name or port number, used with file DSN or in a DSN-less connection.

db2cli.ini keyword syntax:

ServiceName = *service name* | *port number*

Default setting:

None

Only applicable when:

Protocol set to TCPIP

Usage notes:

Use this option in conjunction with the Hostname option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the PROTOCOL option is set to TCPIP.

Specify either the server system's service name or its port number. The service name must be available for lookup at the client machine if it is used.

Related concepts:

- "db2cli.ini initialization file" on page 277

Related reference:

- "CLI/ODBC configuration keywords listing by category" on page 279
- "Hostname CLI/ODBC configuration keyword" on page 327
- "Protocol CLI/ODBC configuration keyword" on page 356

SkipTrace CLI/ODBC configuration keyword

Keyword description:

Excludes CLI trace information from the DB2 trace.

db2cli.ini keyword syntax:

SkipTrace = 0 | 1

Default setting:

Do not skip the trace function.

Usage notes:

This keyword can improve performance by allowing the DB2 trace function to bypass CLI applications. Therefore, if the DB2 trace facility db2trc is turned on and this keyword is set to 1, the trace will not contain information from the execution of the CLI application.

Turning SkipTrace on is recommended for production environments on the UNIX platform where trace information is not required. Test environments may benefit, however, from having trace output, so this keyword can be turned off (or left at its Default setting) when detailed execution information is desired.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- "db2cli.ini initialization file" on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “db2trc - Trace command” in *Command Reference*

SQLOverrideFileName CLI/ODBC configuration keyword

Keyword description:

Specify the location of the override file, which lists CLI statement attribute settings for particular SQL statements.

db2cli.ini keyword syntax:

SQLOverrideFileName = <absolute or relative path name>

Default setting:

No override file is used.

Usage notes:

The SQLOverrideFileName keyword specifies the location of the override file to be read by the DB2 CLI driver. An override file contains values for CLI statement attributes that apply to particular SQL statements. Any of the supported statement attributes can be specified. The following is an example of an override file containing attribute settings specific to two SQL statements:

```
[Common]
Stmts=2
```

```
[1]
StmtIn=SELECT * FROM Employee
StmtAttr=SQL_ATTR_BLOCK_FOR_ROWS=50;SQL_ATTR_OPTIMIZE_FOR_ROWS=1;
```

```
[2]
StmtIn=SELECT * FROM Sales
StmtAttr=SQL_ATTR_MAX_ROWS=25;
```

The number specified by “Stmts” in the “[Common]” section of the override file equals the number of SQL statements contained in the override file.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

StaticCapFile CLI/ODBC configuration keyword

Keyword description:

Specify the Capture File name and optionally the path where it will be saved.

db2cli.ini keyword syntax:

StaticCapFile = < Full file name >

Default setting:

None - you must specify a capture file.

Only applicable when:

StaticMode is set to Capture or Match

Usage notes:

This keyword is used to specify the Capture File name and optionally the directory where it will be saved.

Related concepts:

- “Capture file for CLI/ODBC/JDBC Static Profiling” on page 187
- “db2cli.ini initialization file” on page 277

Related tasks:

- “Creating static SQL with CLI/ODBC/JDBC Static Profiling” on page 185

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “StaticLogFile CLI/ODBC configuration keyword” on page 369
- “StaticMode CLI/ODBC configuration keyword” on page 370
- “StaticPackage CLI/ODBC configuration keyword” on page 370

StaticLogFile CLI/ODBC configuration keyword

Keyword description:

Specify the Static Profiling Log File name and optionally the directory where it will be saved.

db2cli.ini keyword syntax:

StaticLogFile = < Full file name >

Default setting:

No Static Profiling Log is created. If a filename is specified without a pathname then the current path will be used.

Only applicable when:

StaticMode is set to Capture or Match

Usage notes:

This keyword is used to specify the Static Profiling Log File name and optionally the directory where it will be saved.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related tasks:

- “Creating static SQL with CLI/ODBC/JDBC Static Profiling” on page 185

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “StaticCapFile CLI/ODBC configuration keyword” on page 368
- “StaticMode CLI/ODBC configuration keyword” on page 370
- “StaticPackage CLI/ODBC configuration keyword” on page 370

StaticMode CLI/ODBC configuration keyword

Keyword description:

Specify whether the CLI/ODBC application will capture SQL or use a static SQL Package for this DSN.

db2cli.ini keyword syntax:

StaticMode = DISABLED | CAPTURE | MATCH

Default setting:

Disabled - SQL statements are not captured and no static SQL package is used.

Usage notes:

This option allows you to specify how the SQL issued by the CLI/ODBC application for this DSN will be processed:

- **DISABLED** = Static mode disabled. No special processing. The CLI/ODBC statements will be executed as dynamic SQL with no change. This is the default.
- **CAPTURE** = Capture Mode. Execute the CLI/ODBC statements as dynamic SQL. If the SQL statements are successful, they will be captured into a file (known as the Capture File) to be bound by the DB2CAP command later.
- **MATCH** = Match mode. Execute the CLI/ODBC statements as static SQL statements if a matching statement is found in the Capture Files specified in StaticPackage. The Capture File must first be bound by the DB2CAP command.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related tasks:

- “Creating static SQL with CLI/ODBC/JDBC Static Profiling” on page 185

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “StaticCapFile CLI/ODBC configuration keyword” on page 368
- “StaticLogFile CLI/ODBC configuration keyword” on page 369
- “StaticPackage CLI/ODBC configuration keyword” on page 370

StaticPackage CLI/ODBC configuration keyword

Keyword description:

Specify the package to be used with the static profiling feature.

db2cli.ini keyword syntax:

StaticPackage = *collection_id.package_name*

Default setting:

None - you must specify a package name.

Only applicable when:

STATICMODE is set to CAPTURE

Usage notes:

This keyword is used to specify the package to be used when the application runs in Match Mode. You first need to use Capture Mode to create the Capture File.

Only the first 7 characters of the indicated package name will be used. A one-byte suffix will be added to represent each isolation level, as follows:

- 0 for Uncommitted Read (UR)
- 1 for Cursor Stability (CS)
- 2 for Read Stability (RS)
- 3 for Repeatable Read (RR)
- 4 for No Commit (NC)

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Isolation levels” in *SQL Reference, Volume 1*

Related tasks:

- “Creating static SQL with CLI/ODBC/JDBC Static Profiling” on page 185

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “StaticCapFile CLI/ODBC configuration keyword” on page 368
- “StaticLogFile CLI/ODBC configuration keyword” on page 369
- “StaticMode CLI/ODBC configuration keyword” on page 370

StreamGetData CLI/ODBC configuration keyword

Keyword description:

Optimize data output stream for SQLGetData() function.

db2cli.ini keyword syntax:

StreamGetData = 0 | 1

Default setting:

DB2 CLI buffers all the data on the client.

Equivalent connection or statement attribute:

SQL_ATTR_STREAM_GETDATA

Usage notes:

The StreamGetData keyword is ignored if Dynamic Data Format is not supported by the server. For applications that do not need to buffer data and are querying data on a server that supports Dynamic Data Format, specify 1 to indicate that data buffering is not required. The DB2 CLI client will optimize the data output stream.

If StreamGetData is set to 1 and DB2 CLI cannot determine the number of bytes still available to return in the output buffer, SQLGetData() returns SQL_NO_TOTAL (-4) as the length when truncation occurs. Otherwise, SQLGetData() returns the number of bytes still available.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLGetData function (CLI) - Get data from a column” in *Call Level Interface Guide and Reference, Volume 2*
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

StreamPutData CLI/ODBC configuration keyword

Keyword description:

Improve performance for data passed through SQLPutData() function calls on one statement handle, by writing data directly to the internal connection-level communication buffer.

db2cli.ini keyword syntax:

StreamPutData = 0 | 1

Default setting:

Do not write data directly to the connection-level buffer; write to the default statement-level buffer instead.

Usage notes:

By default, DB2 CLI writes data passed in through SQLPutData() function calls to an internal statement-level buffer. On the subsequent SQLParamData() call, the contents of the buffer are then written to an internal connection-level communication buffer and sent to the server. If only one statement handle is used to insert data into a target database on a particular connection at a given point in time, then you can improve performance by setting StreamPutData=1. This causes DB2 CLI to write the put data directly to the connection-level buffer. If, however, multiple statements concurrently insert data into a target database on a particular connection, then setting StreamPutData=1 may decrease performance and result in unexpected application errors, as the statements in the shared connection-level communication buffer will be prone to serialization.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLParamData function (CLI) - Get next parameter for which a data value is needed” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLPutData function (CLI) - Passing data value for a parameter” in *Call Level Interface Guide and Reference, Volume 2*

SysSchema CLI/ODBC Configuration Keyword

Keyword description:

Set an alternative schema to be searched in place of the SYSIBM schema.

db2cli.ini keyword syntax:

SysSchema = **alternative schema**

Default setting:

The default table qualifier name used when querying DB2 for z/OS is SYSIBM.

Usage notes:

This option indicates an alternative schema, or table qualifier, to be searched in place of the SYSIBM schema when the DB2 CLI and ODBC Catalog Function calls are issued to obtain system catalog information from DB2 for z/OS.

Using this new schema name, the system administrator can define a set of views, or a copies of the tables, consisting of a subset of the rows for system catalog tables such as:

- SYSIBM.SYSCOLAUTH
- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSDATATYPES
- SYSIBM.SYSFOREIGNKEYS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSKEYS
- SYSIBM.SYSKEYCOLUSES
- SYSIBM.SYSPARMS
- SYSIBM.SYSRELS
- SYSIBM.SYSROUTINES
- SYSIBM.SYSTABAUTH
- SYSIBM.SYSTABCONST
- SYSIBM.SYSTABLES
- SYSIBM.SYSSYNONYMS

For example, if the set of views, or a copies tables, for the system catalog tables is in the ACME schema, then the view (or copy of the table) for SYSIBM.SYSTABLES is ACME.SYSTABLES; and SysSchema should be set to ACME.

For applications that automatically query the system catalogs for all table names, defining and using limited views of the system catalog tables reduces the number of tables listed by the application. This can reduce the time it takes for the application to query table information since a subset of table names is returned.

Defining and using copies of the system catalog tables, with the same indexes defined on the copy as those defined on the system table, can reduce the time it takes for applications to query the database.

The SchemaList, TableType and DBName keywords can be used in conjunction with the SysSchema keyword to further limit the number of tables for which information is returned.

For DB2 for z/OS Version 7 with PTF UK08673 (contains APAR PK05102), and DB2 for z/OS Version 8 with PTF UK08674 (contains APAR PK05102), the keywords SysSchema and CLISchema have the same functionality. When SysSchema and CLISchema are both set, CLISchema is ignored.

For more information about which system catalog tables can be used with SysSchema, and about the function of SysSchema, refer to the documentation for APAR PK05102 by visiting:

Support for IBM mainframes

and searching for "PK05102".

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

TableType CLI/ODBC configuration keyword

Keyword description:

Define a default list of TABLETYPES returned when querying table information.

db2cli.ini keyword syntax:

```
TableType = " 'TABLE' | , 'ALIAS' | , 'VIEW' | , 'INOPERATIVE  
VIEW' | , 'SYSTEM TABLE' | , 'SYNONYM' "
```

Default setting:

No default list of TABLETYPES is defined.

Usage notes:

If there is a large number of tables defined in the database, a tabletype string can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application.

Any number of the values can be specified. Each type must be delimited with single quotes, separated by commas, and in uppercase. The entire string must also be enclosed in double quotes. For example:

```
TableType="'TABLE', 'VIEW'"
```

This option can be used in conjunction with DBNAME and SCHEMALIST to further limit the number of tables for which information will be returned.

TableType is used to provide a default for the DB2 CLI function that retrieves the list of tables, views, aliases, and synonyms in the database. If the application does not specify a table type on the function call, and this keyword is not used, information about all table types is returned. If the application does supply a value for the *tabletype* on the function call, then that argument value will override this keyword value.

If TableType includes any value other than TABLE, then the DBName keyword setting cannot be used to restrict information to a particular DB2 for z/OS database.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “DBName CLI/ODBC configuration keyword” on page 314
- “SchemaList CLI/ODBC configuration keyword” on page 365

TempDir CLI/ODBC configuration keyword

Keyword description:

Define the directory used for temporary files.

db2cli.ini keyword syntax:

TempDir = < full path name >

Default setting:

Use the system temporary directory specified by the TEMP or TMP environment variables.

Usage notes:

When working with Large Objects (CLOBS, BLOBS, etc...), when data conversion occurs, or when data is sent to the server in pieces, a temporary file is often created on the client machine to store the information. Using this option you can specify a location for these temporary files. The system temporary directory will be used if nothing is specified.

The keyword is placed in the data source specific section of the db2cli.ini file, and has the following syntax:

- TempDir= F:\DB2TEMP

The path specified must already exist and the user executing the application must have the appropriate authorities to write files to it. When the DB2 CLI Driver attempts to create temporary files, an SQLSTATE of HY507 will be returned if the path name is invalid, or if the temporary files cannot be created in the directory specified.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Large object usage in CLI applications” on page 96

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

Trace CLI/ODBC configuration keyword

Keyword description:

Turn on the DB2 CLI/ODBC trace facility.

db2cli.ini keyword syntax:

Trace = 0 | 1

Default setting:

No trace information is captured.

Equivalent environment attribute:

SQL_ATTR_TRACE

Usage notes:

When this option is on (1), CLI/ODBC trace records are appended to the file indicated by the TraceFileName configuration parameter or to files in the subdirectory indicated by the TracePathName configuration parameter. Trace will have no effect if neither TraceFileName or TracePathName is set.

The TraceRefreshInterval keyword sets the interval in seconds at which the Trace keyword is read from the db2cli.ini file. This allows you to dynamically turn off the CLI/ODBC trace within n seconds.

For example, to set up a CLI/ODBC trace file that is written to disk after each trace entry:

```
[COMMON]
Trace=1
TraceFileName=E:\TRACES\CLI\MONDAY.CLI
TraceFlush=1
```

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Environment attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381
- “TraceRefreshInterval CLI/ODBC configuration keyword” on page 383

TraceComm CLI/ODBC configuration keyword

Keyword description:

Include information about each network request in the trace file.

db2cli.ini keyword syntax:

TraceComm = 0 | 1

Default setting:

0 - No network request information is captured.

Only applicable when:

the CLI/ODBC Trace option is turned on.

Usage notes:

When TraceComm is set on (1) then the following information about each network request will be included in the trace file:

- which DB2 CLI functions are processed completely on the client and which DB2 CLI functions involve communication with the server
- the number of bytes sent and received in each communication with the server
- the time spent communicating data between the client and server

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381

TraceErrImmediate CLI/ODBC configuration keyword

Keyword description:

Write diagnostic records to the CLI/ODBC trace when they are generated.

db2cli.ini keyword syntax:

TraceErrImmediate = 0 | 1

Default setting:

Diagnostic records are only written to the trace file when SQLGetDiagField() or SQLGetDiagRec() is called; or “Unretrieved Error Message” is written to the trace file for handles which had diagnostic records that were left unretrieved.

Only applicable when:

the CLI/ODBC Trace option is turned on.

Usage notes:

Setting TraceErrImmediate=1 helps in determining when errors occur during application execution by writing diagnostic records to the CLI/ODBC trace file at the time the records are generated. This is especially useful for applications that do not retrieve diagnostic information using SQLGetDiagField() and SQLGetDiagRec(), because the diagnostic records that were generated on a handle will be lost if they are not retrieved or written to the trace file before the next function is called on the handle.

If TraceErrImmediate=0 (the default setting), then diagnostic records will only be written to the trace file if an application calls SQLGetDiagField() or SQLGetDiagRec() to retrieve diagnostic information. If the application does not retrieve diagnostic information through function calls and this keyword is set to 0, then the “Unretrieved Error Message” entry will be written to the trace file if a diagnostic record exists, when a function is next called on the handle.

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFlushOnError CLI/ODBC configuration keyword” on page 379

TraceFileName CLI/ODBC configuration keyword

Keyword description:

File to which all DB2 CLI/ODBC trace information is written.

db2cli.ini keyword syntax:

TraceFileName = < **fully qualified file name** >

Default setting:

None

Only applicable when:

the Trace option is turned on.

Usage notes:

If the file specified does not exist, then it will be created; otherwise, the new trace information will be appended to the end of the file. However, the path the file is expected in must exist.

If the filename given is invalid or if the file cannot be created or written to, no trace will occur and no error message will be returned.

This option is only used when the Trace option is turned on. This will be done automatically when you set this option in the CLI/ODBC Configuration utility.

The TracePathName option will be ignored if this option is set.

DB2 CLI trace should only be used for debugging purposes. It will slow down the execution of the CLI/ODBC driver, and the trace information can grow quite large if it is left on for extended periods of time.

The TraceFileName keyword option should not be used with multi-process or multithreaded applications as the trace output for all threads or processes will be written to the same log file, and the output for each thread or process will be difficult to decipher. Furthermore, semaphores are used to control access to the shared trace file which could change the behavior of multithreaded applications. There is no default DB2 CLI trace output log file name.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381

TraceFlush CLI/ODBC configuration keyword

Keyword description:

Force a write to disk after n CLI/ODBC trace entries.

db2cli.ini keyword syntax:

TraceFlush = 0 | **positive integer**

Default setting:

Do not write after every entry.

Only applicable when:

the CLI/ODBC Trace option is turned on.

Usage notes:

TraceFlush specifies how often trace information is written to the CLI trace file. By default, TraceFlush is set to 0 and each DB2 CLI trace file is kept open until the traced application or thread terminates normally. If the application terminates abnormally, some trace information that was not written to the trace log file may be lost.

Set this keyword to a positive integer to force the DB2 CLI driver to close and re-open the appropriate trace file after the specified number of trace entries. The smaller the value of the TraceFlush keyword, the greater the impact DB2 CLI tracing has on the performance of the application. Setting TraceFlush=1 has the most impact on performance, but will ensure that each entry is written to disk before the application continues to the next statement.

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TracePathName CLI/ODBC configuration keyword” on page 381

TraceFlushOnError CLI/ODBC configuration keyword

Keyword description:

Write all CLI/ODBC trace entries to disk when an error occurs.

db2cli.ini keyword syntax:

TraceFlushOnError = 0 | **1**

Default setting:

Do not write CLI/ODBC trace entries as soon as an error occurs.

Only applicable when:

the CLI/ODBC Trace option is turned on.

Usage notes:

Setting TraceFlushOnError=1 forces the DB2 CLI driver to close and re-open the trace file each time an error is encountered. If TraceFlushOnError is left at its default value of 0, then trace file will only be closed when the application terminates normally or the interval specified by the TraceFlush keyword is reached. If the application process were to terminate abnormally when TraceFlushOnError=0, then valuable trace information may be lost. Setting TraceFlushOnError=1 may impact performance, but will ensure that trace entries associated with errors are written to disk.

This option is only used when the Trace CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “TraceErrImmediate CLI/ODBC configuration keyword” on page 377
- “TraceFlush CLI/ODBC configuration keyword” on page 379

TraceLocks CLI/ODBC configuration keyword

Keyword description:

Only trace lock timeouts in the CLI/ODBC trace.

db2cli.ini keyword syntax:

TraceLocks = 0 | 1

Default setting:

Trace information is not limited to only lock timeouts.

Only applicable when:

the Trace option is turned on.

Usage notes:

When TraceLocks is set to 1, lock timeouts will be recorded in the trace file.

This option is only used when the CLI/ODBC TRACE option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381

TracePathName CLI/ODBC configuration keyword

Keyword description:

Subdirectory used to store individual DB2 CLI/ODBC trace files.

db2cli.ini keyword syntax:

TracePathName = < **fully qualified subdirectory name** >

Default setting:

None

Only applicable when:

the Trace option is turned on.

Not applicable when:

the TraceFileName option is turned on.

Usage notes:

Each thread or process that uses the same DLL or shared library will have a separate DB2 CLI/ODBC trace file created in the specified directory. A concatenation of the application process ID and the thread sequence number is automatically used to name trace files.

No trace will occur, and no error message will be returned, if the subdirectory given is invalid or if it cannot be written to.

This option is only used when the Trace option is turned on. This will be done automatically when you set this option in the CLI/ODBC Configuration utility.

It will be ignored if the DB2 CLI/ODBC option TraceFileName is used.

DB2 CLI trace should only be used for debugging purposes. It will slow down the execution of the CLI/ODBC driver, and the trace information can grow quite large if it is left on for extended periods of time.

If both TraceFileName and TracePathName are specified, the TraceFileName keyword takes precedence and TracePathName will be ignored.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378

- “TraceFlush CLI/ODBC configuration keyword” on page 379

TracePIDList CLI/ODBC configuration keyword

Keyword description:

Restrict the process IDs for which the CLI/ODBC trace will be enabled.

db2cli.ini keyword syntax:

TracePIDList = <no value specified> | <comma-delimited list of process IDs>

Default setting:

All of the process IDs will be traced when the CLI/ODBC trace is run.

Usage notes:

Use this keyword for applications that create many processes. Capturing the CLI/ODBC trace for such applications can generate many trace files. By using this keyword you can collect the trace of specific problematic processes of applications.

If no value is specified for this keyword, all process IDs will be traced. Otherwise, specify a comma-delimited list of process IDs which you want to be traced when the CLI/ODBC trace runs.

The TraceRefreshInterval keyword must be set to some value before initializing your application, otherwise, the TracePIDList keyword will not take effect.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2 databases.)

To use the TracePIDList keyword:

1. Ensure the Trace CLI/ODBC keyword is set to zero or is not specified in the db2cli.ini file.
2. Add the TraceRefreshInterval CLI/ODBC keyword to the Common section of the db2cli.ini file as follows:

```
[COMMON]
TraceRefreshInterval=<some positive integer>
```

3. Start your application.
4. Using an operating system command such as **ps** (on a UNIX and Linux-based operating systems), determine the process IDs of the processes that you want to collect the CLI/ODBC trace for.
5. Turn CLI/ODBC tracing on and add the process IDs identified to the Common section of the db2cli.ini file by including the following keywords:

```
[COMMON]
Trace=1
TracePathName=<fully-qualified subdirectory name>
TracePIDList=<comma-delimited list of process IDs>
```

CLI/ODBC traces containing information of the process IDs specified will be located in the directory specified by the TracePathName keyword.. You might also see extra empty files that can be ignored.

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TracePathName CLI/ODBC configuration keyword” on page 381
- “TracePIDTID CLI/ODBC configuration keyword” on page 383
- “TraceRefreshInterval CLI/ODBC configuration keyword” on page 383
- “UPDATE CLI CONFIGURATION command” in *Command Reference*

TracePIDTID CLI/ODBC configuration keyword

Keyword description:

Capture the process ID and thread ID for each item being traced.

db2cli.ini keyword syntax:

TracePIDTID = 0 | 1

Default setting:

The process ID and thread ID for the trace entries are not captured.

Only applicable when:

the Trace option is turned on.

Usage notes:

When TracePIDTID is set to 1, the process ID and thread ID for each captured item will be recorded in the trace file. This effect is helpful when the Trace keyword is enabled and multiple applications are executing. This is because Trace writes trace information for all executing applications to a single file. Enabling TracePIDTID differentiates the recorded information by process and thread.

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381
- “TracePIDList CLI/ODBC configuration keyword” on page 382

TraceRefreshInterval CLI/ODBC configuration keyword

Keyword description:

Set the interval (in seconds) at which the Trace and TracePIDList keywords are read from the Common section of the db2cli.ini file.

db2cli.ini keyword syntax:

TraceRefreshInterval = 0 | **positive integer**

Default setting:

The Trace and TracePIDList keywords will only be read from the db2cli.ini file when the application is initialized.

Usage notes:

Setting this keyword before an application is initialized allows you to turn off dynamically the CLI/ODBC trace within n seconds.

Note: Setting TraceRefreshInterval while the application is running will no effect. For this keyword to take effect, it must be set before the application is initialized.

Only the Trace and TracePIDList keywords will be refreshed from the db2cli.ini file if this keyword is set. No other CLI/ODBC configuration keywords will be re-read.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TracePIDList CLI/ODBC configuration keyword” on page 382

TraceStmtOnly CLI/ODBC configuration keyword

Keyword description:

Only trace dynamic SQL statements in the CLI/ODBC trace.

db2cli.ini keyword syntax:

TraceStmtOnly = 0 | **1**

Default setting:

Trace information is not limited to only dynamic SQL statements.

Only applicable when:

the Trace option is turned on.

Usage notes:

When TraceStmtOnly is set to 1, only dynamic SQL statements will be recorded in the trace file.

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381

TraceTime CLI/ODBC configuration keyword

Keyword description:

Capture elapsed time counters in the trace file.

db2cli.ini keyword syntax:

TraceTime = 1 | 0

Default setting:

Elapsed time counters are included in the trace file.

Only applicable when:

the Trace option is turned on.

Usage notes:

When TraceTime is set to 1, elapsed time counters will be captured in the trace file. For example:

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
  —> Time elapsed - +6.785751E+000 seconds ( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <— SQL_SUCCESS Time elapsed - +2.527400E-002 seconds
```

Turn TraceTime off, by setting it to 0, to improve performance or to generate smaller trace files. For example:

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <— SQL_SUCCESS
```

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379

- “TracePathName CLI/ODBC configuration keyword” on page 381
- “TraceTimestamp CLI/ODBC configuration keyword” on page 386

TraceTimestamp CLI/ODBC configuration keyword

Keyword description:

Capture different types of timestamp information in the CLI/ODBC trace.

db2cli.ini keyword syntax:

TraceTimestamp = 0 | 1 | 2 | 3

Default setting:

No timestamp information is written to the trace file.

Only applicable when:

the Trace option is turned on.

Usage notes:

Setting TraceTimeStamp to a value other than the default of 0 means the current timestamp or absolute execution time is added to the beginning of each line of trace information as it is being written to the DB2 CLI trace file. The following settings indicate what type of timestamp information is captured in the trace file:

- 0 = no timestamp information
- 1 = processor ticks and ISO timestamp (absolute execution time in seconds and milliseconds, followed by a timestamp)
- 2 = processor ticks (absolute execution time in seconds and milliseconds)
- 3 = ISO timestamp

This option is only used when the CLI/ODBC Trace option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “CLI/ODBC/JDBC trace facility” on page 189
- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Trace CLI/ODBC configuration keyword” on page 375
- “TraceFileName CLI/ODBC configuration keyword” on page 378
- “TraceFlush CLI/ODBC configuration keyword” on page 379
- “TracePathName CLI/ODBC configuration keyword” on page 381
- “TraceTime CLI/ODBC configuration keyword” on page 385

Trusted_Connection CLI/ODBC configuration keyword

Keyword description:

Allow a connection to be made with the current authenticated user.

Syntax:

Trusted_Connection=Yes

Note: This keyword will have no effect if set in the `db2cli.ini` file. It should instead be provided in the connection string to `SQLDriverConnect()`.

Default setting:

DB2 CLI uses the user ID and password information provided in the connection string to `SQLDriverConnect()`, not the current authenticated user.

Usage notes:

DB2 CLI applications that connect to a database will typically connect using the function `SQLDriverConnect()`. One of the input arguments for this function is the *DriverCompletion* value, which determines when a dialog will be opened. The following are the values of *DriverCompletion* :

- `SQL_DRIVER_PROMPT`: A dialog is always initiated.
- `SQL_DRIVER_COMPLETE`: A dialog is only initiated if there is insufficient information in the connection string.
- `SQL_DRIVER_COMPLETE_REQUIRED`: A dialog is only initiated if there is insufficient information in the connection string. Only mandatory information is requested. The user is prompted for required information only.
- `SQL_DRIVER_NOPROMPT`: The user is not prompted for any information. A connection is attempted with the information contained in the connection string. If there is not enough information, `SQL_ERROR` is returned.

Note: More details on *DriverCompletion* can be found in the documentation for `SQLDriverConnect()`.

Some applications, for example, those in a Kerberos environment, might require that a user be able to connect to a DB2 server without providing a user ID or password. If the application uses the `SQL_DRIVER_NO_PROMPT` option on the `SQLDriverConnect()` call, the connection will be attempted without the user authentication. This keyword is then not required.

In the case where a third party application is involved and the prompt level used by the application is something other than `SQL_DRIVER_NO_PROMPT`, DB2 CLI will open a dialog to request the missing information from the user. Setting `Trusted_Connection` to `Yes`, by providing it to the input connection string for `SQLDriverConnect()` ("`Trusted_Connection=Yes`"), causes DB2 CLI to ignore any user ID or password string (including blank strings) from the connection string and ignore the prompt level of the connection function. DB2 CLI will use the current authenticated user to attempt the connection to the database. If the connection attempt fails, the user will be prompted for the user ID and password.

This keyword is used only in the connection string for `SQLDriverConnect()`; setting it in the `db2cli.ini` file will have no effect.

Related concepts:

- "db2cli.ini initialization file" on page 277
- "Kerberos authentication details" in *Administration Guide: Implementation*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” in *Call Level Interface Guide and Reference, Volume 2*

TxnIsolation CLI/ODBC configuration keyword

Keyword description:

Set the default isolation level.

db2cli.ini keyword syntax:

TxnIsolation = 1 | 2 | 4 | 8 | 32

Default setting:

Read Committed (Cursor Stability)

Only applicable when:

the default isolation level is used. This keyword will have no effect if the application has specifically set the isolation level.

Equivalent statement attribute:

SQL_ATTR_TXN_ISOLATION

Usage notes:

Sets the isolation level to:

- 1 = SQL_TXN_READ_UNCOMMITTED - Read Uncommitted (Uncommitted read)
- 2 = SQL_TXN_READ_COMMITTED (default) - Read Committed (Cursor stability)
- 4 = SQL_TXN_REPEATABLE_READ - Repeatable Read (Read Stability)
- 8 = SQL_TXN_SERIALIZABLE - Serializable (Repeatable read)
- 32 = SQL_TXN_NOCOMMIT - (No Commit, DB2 Universal Database for AS/400 only; this is similar to autocommit)

The words in parentheses are IBM’s terminology for the equivalent SQL92 isolation levels. Note that *no commit* is not an SQL92 isolation level and is supported only on DB2 Universal Database for AS/400.

This keyword is only applicable if the default isolation level is used. If the application specifically sets the isolation level for a connection or statement handle, then this keyword will have no effect on that handle.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “Isolation levels” in *SQL Reference, Volume 1*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “Statement attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

UID CLI/ODBC configuration keyword

Keyword description:

Define default user ID.

db2cli.ini keyword syntax:

UID = *userid*

Default setting:

None

Usage notes:

The specified *userid* value is used if a *userid* is not provided by the application at connect time.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “PWD CLI/ODBC configuration keyword” on page 357

Underscore CLI/ODBC configuration keyword

Keyword description:

Specify whether the underscore character ‘_’ is treated as a wildcard.

db2cli.ini keyword syntax:

Underscore = 0 | 1

Default setting:

The underscore character matches any single character or no character.

Usage notes:

This keyword specifies if the underscore character ‘_’ will be recognized as a wildcard or only as the underscore character. The possible settings are as follows:

- 0 - The underscore character is treated only as the underscore character.
- 1 - The underscore character is treated as a wildcard that matches any single character, including no character.

Setting Underscore to 0 can improve performance when there are database objects with names that contain underscores.

This keyword applies only to the following catalog functions that accept search patterns as arguments:

- SQLColumnPrivileges()
- SQLColumns()
- SQLProcedureColumns()
- SQLProcedures()
- SQLTablePrivileges()
- SQLTables()

Note that catalog functions may only accept search patterns on particular arguments. Refer to the documentation of the specific function for details.

Related concepts:

- “db2cli.ini initialization file” on page 277

- “Input arguments on catalog functions in CLI applications” on page 168

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLColumns function (CLI) - Get column information for a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLProcedures function (CLI) - Get list of procedure names” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLTablePrivileges function (CLI) - Get privileges associated with a table” in *Call Level Interface Guide and Reference, Volume 2*
- “SQLTables function (CLI) - Get table information” in *Call Level Interface Guide and Reference, Volume 2*

UseOldStpCall CLI/ODBC configuration keyword

Keyword description:

Controls how cataloged procedures are invoked.

db2cli.ini keyword syntax:

UseOldStpCall = 0 | 1

Default setting:

Invokes procedures using the new CALL method where GRANT EXECUTE must be granted on the procedure.

Usage notes:

Prior to DB2 UDB Version 8, the invoker of a procedure had to have EXECUTE privilege on any package invoked from the procedure. Now, the invoker must have EXECUTE privilege on the procedure and only the definer of the procedure has to have EXECUTE privilege on any required packages.

This keyword controls which method is used to invoke the procedure. Setting UseOldStpCall on causes the procedure to be invoked using the deprecated sqlproc() API when the precompiler fails to resolve a procedure on a CALL statement. Turning this keyword off will invoke procedures where GRANT EXECUTE must be granted on the procedure.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related tasks:

- “Calling stored procedures from CLI applications” on page 113

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

UseServerMsgSP CLI/ODBC configuration keyword

Keyword description:

Use the DB2 for z/OS DSNACCMG server stored procedure to retrieve message text from DB2 for z/OS.

db2cli.ini keyword syntax:

UseServerMsgSP = 0 | 1

Default setting:

DB2 CLI does not use the server stored procedure to return messages, but uses the local message files.

Equivalent connection attribute:

SQL_ATTR_SERVER_MSGTXT_SP

Usage notes:

Applications using this keyword should also set the ServerMsgMask CLI/ODBC configuration keyword to indicate when DB2 CLI should call this procedure to retrieve the message information from the server. If the ServerMsgMask CLI/ODBC configuration keyword is not set, then the default is to check the local message files first. See the ServerMsgMask CLI/ODBC configuration keyword for more details on the options available.

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “ServerMsgMask CLI/ODBC configuration keyword” on page 366
- “Connection attributes (CLI) list” in *Call Level Interface Guide and Reference, Volume 2*

WarningList CLI/ODBC configuration keyword

Keyword description:

Specify which errors to downgrade to warnings.

db2cli.ini keyword syntax:

WarningList = " 'xxxxx', 'yyyyy', ..."

Default setting:

Do not downgrade any SQLSTATEs.

Usage notes:

Any number of SQLSTATEs returned as errors can be downgraded to warnings. Each must be delimited with single quotes, separated by commas, and in uppercase. The entire string must also be enclosed in double quotes. For example:

```
WarningList=" '01S02', 'HY090' "
```

Related concepts:

- “db2cli.ini initialization file” on page 277

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279
- “IgnoreWarnList CLI/ODBC configuration keyword” on page 328

XMLDeclaration CLI/ODBC configuration keyword

Keyword description:

Controls the generation of an XML declaration when XML data is implicitly serialized to an application variable.

db2cli.ini keyword syntax:

XMLDeclaration = **non-negative integer** < 7 | 7

Default setting:

A BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.

Usage notes:

The XMLDeclaration keyword controls which elements of an XML declaration are prepended to an application buffer when XML data is implicitly serialized to an application buffer. This setting does not affect the result of the XMLSERIALIZE function.

The following values represent components to be generated during implicit serialization. Set this keyword by adding together the value of each component required.

- | | |
|---|--|
| 0 | No declarations or byte order marks (BOMs) are added to the output buffer. |
| 1 | A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16 or UTF-32. (Although a UTF-8 BOM exists, the database server does not generate it, even if the target encoding is UTF-8.) |
| 2 | A minimal XML declaration is generated, containing only the XML version. |
| 4 | An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this keyword. |

For example, if you wanted a BOM and minimal XML declaration (without an encoding attribute) to be generated during implicit serialization, you would set XMLDeclaration = 3, where 3 is the sum of 1 (the value to indicate generation of a BOM) and 2 (the value to indicate generation of a minimal XML declaration).

To prevent any declarations or BOM from being generated, set XMLDeclaration as follows: XMLDeclaration = 0.

Related concepts:

- “db2cli.ini initialization file” on page 277
- “XML data encoding” in *XML Guide*
- “XML data handling in CLI applications - Overview” on page 117
- “XML data type” in *XML Guide*

Related reference:

- “CLI/ODBC configuration keywords listing by category” on page 279

- “XMLSERIALIZE scalar function” in *SQL Reference, Volume 1*

Part 6. Data conversion

Chapter 36. Data conversion

Data conversions supported in CLI 397 C to SQL data conversion in CLI 406
 SQL to C data conversion in CLI 399

This chapter describes the data conversions supported in CLI, along with the details of SQL to C and C to SQL data type conversions.

Data conversions supported in CLI

The table below shows which data type conversions are supported by DB2 CLI.

The first column contains the data type of the SQL data type. The remaining columns represent the C data types. If the C data type columns contain:

- D** The conversion is supported and this is the default conversion for the SQL data type.
- X** all IBM DBMSs support the conversion.
- blank** no IBM DBMS supports the conversion.

As an example, the table indicates that an SQLCHAR (or a C character) string can be converted into an SQL_C_LONG (a signed long). In contrast, an SQLINTEGER cannot be converted to an SQL_C_DBCHAR.

Refer to the tables of data type attributes (precision, scale, length, and display) for more information on the data type formats.

Table 27. Supported data conversions

SQL data type	SQL_C_CHAR	SQL_C_WCHAR	SQL_C_DBC	SQL_C_WDBC	SQL_C_NUMERIC	SQL_C_INTEGER	SQL_C_REAL	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_LONG	SQL_C_SLONG	SQL_C_SHORT	SQL_C_USHORT	SQL_C_INT8	SQL_C_UINT8	SQL_C_INT16	SQL_C_UINT16	SQL_C_INT32	SQL_C_UINT32	SQL_C_TIMESTAMP	SQL_C_DATE	SQL_C_TIME	SQL_C_BINARY	SQL_C_DBCHAR	SQL_C_WDBCHAR	SQL_C_DBCHAR2	SQL_C_WDBCHAR2
SQL_CHAR	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
SQL_WCHAR ⁴	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
SQL_VARCHAR	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
SQL_WVARCHAR ⁴	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					
SQL_LONG VARCHAR	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X					

Table 27. Supported data conversions (continued)

SQL data type	SQL_CHAR	SQL_VARCHAR	SQL_CLOB	SQL_BLOB	SQL_DBCLOB	SQL_CLOB_LOCATOR	SQL_BLOB_LOCATOR	SQL_DBCLOB_LOCATOR	SQL_NUMERIC	SQL_DECIMAL	SQL_DECFLOAT	SQL_INTEGER	SQL_SMALLINT	SQL_FLOAT	SQL_DOUBLE	SQL_REAL	SQL_BIGINT	SQL_TINYINT	SQL_BIT	SQL_TYPE_DATE	SQL_TYPE_TIME	SQL_CLOB	SQL_BLOB	SQL_DBCLOB	SQL_CLOB_LOCATOR	SQL_BLOB_LOCATOR	SQL_DBCLOB_LOCATOR	
SQL_BINARY	X	X																										
SQL_VARBINARY	X	X																										
SQL_LONG_VARBINARY	X	X																										
SQL_GRAPHIC	X	X	D						X	X	X	X	X	X	X	X	X	X						X				
SQL_VARGRAPHIC	X	X	D						X	X	X	X	X	X	X	X	X	X						X				
SQL_LONG_VARGRAPHIC	X	X	D						X	X	X	X	X	X	X	X	X	X						X				
SQL_CLOB	D	X																						X	X			
SQL_BLOB	X	X																						D		X		
SQL_DBCLOB	X	X	D																					X			X	
SQL_CLOB_LOCATOR																									D			
SQL_BLOB_LOCATOR																										D		
SQL_DBCLOB_LOCATOR																											D	
SQL_NUMERIC	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_DECIMAL	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_DECFLOAT	D	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_INTEGER	X	X		X	X	X	X	X	X	X	X	D ³	D ³	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_SMALLINT	X	X		X	X	X	X	D ²	D ²	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_FLOAT	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X
SQL_DOUBLE	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X
SQL_REAL	X	X		X	X	X	X	X	X	X	X	X	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X	X
SQL_BIGINT	X	X		X	X	X	X	X	X	X	X	X	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X	X
SQL_TINYINT	X	X		X	D ¹	D ¹	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_BIT	X	X																									D	
SQL_TYPE_DATE	X	X																			D	X						
SQL_TYPE_TIME	X	X																			D	X						

- the third and fourth columns list the values (for each outcome) of the *rgbValue* and *pcbValue* arguments specified in the `SQLBindCol()` or `SQLGetData()` after the driver has attempted to convert the data.
- the last column lists the `SQLSTATE` returned for each outcome by `SQLFetch()`, `SQLExtendedFetch()`, `SQLGetData()` or `SQLGetSubString()`.

The tables list the conversions defined by ODBC to be valid for a given SQL data type.

If the *fCType* argument in `SQLBindCol()` or `SQLGetData()` contains a value not shown in the table for a given SQL data type, `SQLFetch()`, or `SQLGetData()` returns the `SQLSTATE 07006` (Restricted data type attribute violation).

If the *fCType* argument contains a value shown in the table but which specifies a conversion not supported by the driver, `SQLFetch()`, or `SQLGetData()` returns `SQLSTATE HYC00` (Driver not capable).

Though it is not shown in the tables, the *pcbValue* argument contains `SQL_NULL_DATA` when the SQL data value is `NULL`. For an explanation of the use of *pcbValue* when multiple calls are made to retrieve data, see `SQLGetData()`.

When SQL data is converted to character C data, the character count returned in *pcbValue* does not include the null termination byte. If *rgbValue* is a null pointer, `SQLBindCol()` or `SQLGetData()` returns `SQLSTATE HY009` (Invalid argument value).

In the following tables:

Length of data

the total length of the data after it has been converted to the specified C data type (excluding the null termination byte if the data was converted to a string). This is true even if data is truncated before it is returned to the application.

Significant digits

the minus sign (if needed) and the digits to the left of the decimal point.

Display size

the total number of bytes needed to display data in the character format.

Converting character SQL data to C data:

The character SQL data types are:

- SQL_CHAR
- SQL_VARCHAR
- SQL_LONGVARCHAR
- SQL_CLOB

Table 28. Converting character SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Length of data < cbValueMax	Data	Length of data	00000
	Length of data >= cbValueMax	Truncated data	Length of data	01004

Table 28. Converting character SQL data to C data (continued)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	00000
	Length of data > cbValueMax	Truncated data	Length of data	01004
SQL_C_SHORT SQL_C_LONG	Data converted without truncation ^a	Data	Size of the C data type	00000
SQL_C_FLOAT SQL_C_FLOAT SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC ^c	Data converted with truncation, but without loss of significant digits ^a	Data	Size of the C data type	01004
	Conversion of data would result in loss of significant digits ^a	Untouched	Size of the C data type	22003
	Data is not a number ^a	Untouched	Size of the C data type	22005
SQL_C_DATE	Data value is a valid date ^a	Data	6 ^b	00000
	Data value is not a valid date ^a	Untouched	6 ^b	22007
SQL_C_TIME	Data value is a valid time ^a	Data	6 ^b	00000
	Data value is not a valid time ^a	Untouched	6 ^b	22007
SQL_C_TIMESTAMP	Data value is a valid timestamp ^a	Data	16 ^b	00000
	Data value is not a valid timestamp ^a	Untouched	16 ^b	22007

Note:

- ^a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.
- ^b This is the size of the corresponding C data type.
- ^c SQL_C_NUMERIC is only supported on Windows platforms.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

Converting graphic SQL data to C data:

The graphic SQL data types are:

- SQL_GRAPHIC
- SQL_VARGRAPHIC
- SQL_LONGVARGRAPHIC
- SQL_DBCLOB

Table 29. Converting GRAPHIC SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Number of double byte characters * 2 <= cbValueMax	Data	Length of data(octects)	00000
	Number of double byte characters * 2 > cbValueMax	Truncated data, to the nearest even byte that is less than <i>cbValueMax</i> .	Length of data(octects)	01004
SQL_C_DBCHAR	Number of double byte characters * 2 < cbValueMax	Data	Length of data(octects)	00000
	Number of double byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than <i>cbValueMax</i> .	Length of data(octects)	01004

Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.

When converting to floating point values, SQLSTATE 22003 will not be returned if non-significant digits of the resulting value are lost.

Converting numeric SQL data to C data:

The numeric SQL data types are:

- SQL_DECIMAL
- SQL_NUMERIC
- SQL_SMALLINT
- SQL_INTEGER
- SQL_BIGINT
- SQL_REAL
- SQL_FLOAT
- SQL_DOUBLE

Table 30. Converting numeric SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Display size < cbValueMax	Data	Length of data	00000
	Number of significant digits < cbValueMax	Truncated data	Length of data	01004
	Number of significant digits >= cbValueMax	Untouched	Length of data	22003

Table 30. Converting numeric SQL data to C data (continued)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_SHORT SQL_C_LONG	Data converted without truncation ^a	Data	Size of the C data type	00000
SQL_C_FLOAT SQL_C_DOUBLE SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT	Data converted with truncation, but without loss of significant digits ^a	Truncated data	Size of the C data type	01004
SQL_C_SBIGINT SQL_C_NUMERIC ^b	Conversion of data would result in loss of significant digits ^a	Untouched	Size of the C data type	22003

Note:

^a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

^b SQL_C_NUMERIC is only supported on Windows platforms.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

Converting binary SQL data to C data:

The binary SQL data types are:

- SQL_BINARY
- SQL_VARBINARY
- SQL_LONGVARBINARY
- SQL_BLOB

Table 31. Converting binary SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	(Length of data) < <i>cbValueMax</i>	Data	Length of data	N/A
	(Length of data) >= <i>cbValueMax</i>	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data <= <i>cbValueMax</i>	Data	Length of data	N/A
	Length of data > <i>cbValueMax</i>	Truncated data	Length of data	01004

Converting XML SQL data to C data:

The XML SQL data type is:

- SQL_XML

Table 32. Converting XML SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Length of data < <i>cbValueMax</i>	Data	Length of data	00000
	Length of data >= <i>cbValueMax</i>	Truncated data	Length of data	01004

Table 32. Converting XML SQL data to C data (continued)

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	00000
	Length of data > cbValueMax	Truncated data	Length of data	01004
SQL_C_DBCHAR	Number of double-byte characters * 2 < cbValueMax	Data	Length of data	00000
	Number of double-byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than cbValueMax	Length of data	01004
SQL_C_WCHAR	Number of double-byte characters * 2 < cbValueMax	Data	Length of data	00000
	Number of double-byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than cbValueMax	Length of data	01004

Notes:

1. SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.
2. Length of data includes any XML declaration in the target encoding.

Converting date SQL data to C data:

The date SQL data type is:

SQL_DATE

Table 33. Converting date SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 11	Data	10	00000
	cbValueMax < 11	Untouched	10	22003
SQL_C_DATE	None ^a	Data	6 ^b	00000
SQL_C_TIMESTAMP	None ^a	Data ^c	16 ^b	00000

Note:

^a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

^b This is the size of the corresponding C data type.

^c The time fields of the `TIMESTAMP_STRUCT` structure are set to zero.

SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.

When the date SQL data type is converted to the character C data type, the resulting string is in the "yyyy-mm-dd" format.

Converting Time SQL Data to C Data:

The time SQL data type is:
SQL_TIME

Table 34. Converting time SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 9	Data	8	00000
	cbValueMax < 9	Untouched	8	22003
SQL_C_TIME	None ^a	Data	6 ^b	00000
SQL_C_TIMESTAMP	None ^a	Data ^c	16 ^b	00000

Note:

- ^a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.
- ^b This is the size of the corresponding C data type.
- ^c The date fields of the `TIMESTAMP_STRUCT` structure are set to the current system date of the machine that the application is running, and the time fraction is set to zero.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

When the time SQL data type is converted to the character C data type, the resulting string is in the "hh:mm:ss" format.

Converting timestamp SQL data to C data:

The timestamp SQL data type is:
SQL_TIMESTAMP

Table 35. Converting timestamp SQL data to C data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Display size < cbValueMax	Data	Length of data	00000
	19 <= cbValueMax <= Display size	Truncated Data ^b	Length of data	01004
	cbValueMax < 19	Untouched	Length of data	22003
SQL_C_DATE	None ^a	Truncated data ^c	6 ^e	01004
SQL_C_TIME	None ^a	Truncated data ^d	6 ^e	01004
SQL_C_TIMESTAMP	None ^a	Data	16 ^e	00000

Note:

- ^a The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.
- ^b The fractional seconds of the timestamp are truncated.
- ^c The time portion of the timestamp is deleted.
- ^d The date portion of the timestamp is deleted.
- ^e This is the size of the corresponding C data type.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

When the timestamp SQL data type is converted to the character C data type, the resulting string is in the "yyyy-mm-dd hh:mm:ss.ffffff" format (regardless of the precision of the timestamp SQL data type). If an application requires the ISO format, set the CLI/ODBC configuration keyword PATCH2=33.

SQL to C data conversion examples:

Table 36. SQL to C data conversion examples

SQL data type	SQL data value	C data type	cbValue max	rgbValue	SQL STATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef\0 ^a	00000
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde\0 ^a	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56\0 ^a	00000
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234\0 ^a	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	---	22003
SQL_DECIMAL	1234.56	SQL_C_FLOAT	ignored	1234.56	00000
SQL_DECIMAL	1234.56	SQL_C_SHORT	ignored	1234	01004
SQL_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31\0 ^a	00000
SQL_DATE	1992-12-31	SQL_C_CHAR	10	---	22003
SQL_DATE	1992-12-31	SQL_C_TIMESTAMP	ignored	1992,12,31, 0,0,0,0 ^b	00000
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12\0 ^a	00000
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1\0 ^a	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	---	22003

Note:

^a "\0" represents a null termination character.

^b The numbers in this list are the numbers stored in the fields of the `TIMESTAMP_STRUCT` structure.

SQLSTATE 00000 is not returned by `SQLGetDiagRec()`, rather it is indicated when the function returns `SQL_SUCCESS`.

Related concepts:

- "XML data type" in *XML Guide*

Related reference:

- "C data types for CLI applications" on page 43
- "C to SQL data conversion in CLI" on page 406
- "Data conversions supported in CLI" on page 397
- "Patch2 CLI/ODBC configuration keyword" on page 352
- "SQL symbolic and default data types for CLI applications" on page 41

C to SQL data conversion in CLI

For a given C data type:

- the first column of the table lists the legal input values of the `fSqlType` argument in `SQLBindParameter()` or `SQLSetParam()`.

- the second column lists the outcomes of a test, often using the length of the parameter data as specified in the *pcbValue* argument in `SQLBindParameter()` or `SQLSetParam()`, which the driver performs to determine if it can convert the data.
- the third column lists the `SQLSTATE` returned for each outcome by `SQLExecDirect()` or `SQLExecute()`.

The tables list the conversions defined by ODBC to be valid for a given SQL data type.

If the *fSqlType* argument in `SQLBindParameter()` or `SQLSetParam()` contains a value not shown in the table for a given C data type, `SQLSTATE 07006` is returned (Restricted data type attribute violation).

If the *fSqlType* argument contains a value shown in the table but which specifies a conversion not supported by the driver, `SQLBindParameter()` or `SQLSetParam()` returns `SQLSTATE HYC00` (Driver not capable).

If the *rgbValue* and *pcbValue* arguments specified in `SQLBindParameter()` or `SQLSetParam()` are both null pointers, that function returns `SQLSTATE HY009` (Invalid argument value).

Length of data

the total length of the data after it has been converted to the specified SQL data type (excluding the null termination byte if the data was converted to a string). This is true even if data is truncated before it is sent to the data source.

Column length

the maximum number of bytes returned to the application when data is transferred to its default C data type. For character data, the length does not include the null termination byte.

Display size

the maximum number of bytes needed to display data in character form.

Significant digits

the minus sign (if needed) and the digits to the left of the decimal point.

Converting character C data to SQL data:

The character C data type is:

`SQL_C_CHAR`

Table 37. Converting character C data to SQL data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data <= Column length	N/A
	Length of data > Column length	22001
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_BIGINT SQL_REAL SQL_FLOAT SQL_DOUBLE	Data converted without truncation	N/A
	Data converted with truncation, but without loss of significant digits	22001
	Conversion of data would result in loss of significant digits	22003
	Data value is not a numeric value	22005

Table 37. Converting character C data to SQL data (continued)

fSQLType	Test	SQLSTATE
SQL_BINARY	(Length of data) < Column length	N/A
SQL_VARBINARY	(Length of data) >= Column length	22001
SQL_LONGVARBINARY		
SQL_BLOB	Data value is not a hexadecimal value	22005
SQL_DATE	Data value is a valid date	N/A
	Data value is not a valid date	22007
SQL_TIME	Data value is a valid time	N/A
	Data value is not a valid time	22007
SQL_TIMESTAMP	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007
SQL_GRAPHIC	Length of data / 2 <= Column length	N/A
SQL_VARGRAPHIC		
SQL_LONGVARGRAPHIC	Length of data / 2 < Column length	22001
SQL_DBCLOB		
SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

Converting numeric C data to SQL data:

The numeric C data types are:

SQL_C_SHORT
 SQL_C_LONG
 SQL_C_FLOAT
 SQL_C_DOUBLE
 SQL_C_TINYINT
 SQL_C_SBIGINT
 SQL_C_BIT

Table 38. Converting numeric C data to SQL data

fSQLType	Test	SQLSTATE
SQL_DECIMAL	Data converted without truncation	N/A
SQL_NUMERIC		
SQL_SMALLINT	Data converted with truncation, but without loss of significant digits	22001
SQL_INTEGER		
SQL_BIGINT		
SQL_REAL	Conversion of data would result in loss of significant digits	22003
SQL_FLOAT		
SQL_DOUBLE		
SQL_CHAR	Data converted without truncation.	N/A
SQL_VARCHAR	Conversion of data would result in loss of significant digits.	22003

Note: When converting to floating point values, SQLSTATE 22003 will not be returned if non-significant digits of the resulting value are lost.

Converting binary C data to SQL data:

The binary C data type is:

SQL_C_BINARY

Table 39. Converting binary C data to SQL data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Length of data <= Column length	N/A
SQL_LONGVARCHAR SQL_CLOB	Length of data > Column length	22001
SQL_BINARY SQL_VARBINARY	Length of data <= Column length	N/A
SQL_LONGVARBINARY SQL_BLOB	Length of data > Column length	22001
SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

Converting DBCHAR C data to SQL data:

The double byte C data type is:

SQL_C_DBCHAR

Table 40. Converting DBCHAR C data to SQL data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Length of data <= Column length x 2	N/A
SQL_LONGVARCHAR SQL_CLOB	Length of data > Column length x 2	22001
SQL_BINARY SQL_VARBINARY	Length of data <= Column length x 2	N/A
SQL_LONGVARBINARY SQL_BLOB	Length of data > Column length x 2	22001
SQL_XML	Data can be implicitly parsed	(several SQLSTATES can be returned)

Converting date C data to SQL data:

The date C data type is:

SQL_C_DATE

Table 41. Converting date C data to SQL data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= 10	N/A
	Column length < 10	22003
SQL_DATE	Data value is a valid date	N/A
	Data value is not a valid date	22007
SQL_TIMESTAMP ^a	Data value is a valid date	N/A
	Data value is not a valid date	22007

Table 41. Converting date C data to SQL data (continued)

fSQLType	Test	SQLSTATE
Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.		
Note: a, the time component of TIMESTAMP is set to zero.		

Converting time C data to SQL data:

The time C data type is:
SQL_C_TIME

Table 42. Converting time C data to SQL data

fSQLType	Test	SQLSTATE
SQL_CHAR	Column length >= 8	N/A
SQL_VARCHAR	Column length < 8	22003
SQL_TIME	Data value is a valid time	N/A
	Data value is not a valid time	22007
SQL_TIMESTAMP ^a	Data value is a valid time	N/A
	Data value is not a valid time	22007

Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.

Note: a The date component of TIMESTAMP is set to the system date of the machine at which the application is running.

Converting timestamp C data to SQL data:

The timestamp C data type is:
SQL_C_TIMESTAMP

Table 43. Converting timestamp C data to SQL data

fSQLType	Test	SQLSTATE
SQL_CHAR	Column length >= Display size	N/A
SQL_VARCHAR	19 <= Column length < Display size ^a	22001
	Column length < 19	22003
SQL_DATE	Time fields are zero	N/A
	Time fields are non-zero	22008
	Data value does not contain a valid date ^b	22007
SQL_TIME	Fractional seconds fields are zero	N/A
	Fractional seconds fields are non-zero	22008
	Data value does not contain a valid time	22007
SQL_TIMESTAMP	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007

Table 43. Converting timestamp C data to SQL data (continued)

fSQLType	Test	SQLSTATE
Note:		
^a	The fractional seconds of the timestamp are truncated.	
^b	The timestamp_struct must reset the hour, minute, second, and fraction to 0, otherwise SQLSTATE 22007 will be returned.	
SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.		

C to SQL data conversion examples:

Table 44. C to SQL data conversion examples

C data type	C data value	SQL data type	Column length	SQL data value	SQL STATE
SQL_C_CHAR	abcdef\0	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef\0	SQL_CHAR	5	abcde	22001
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	6	1234.56	N/A
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	5	1234.5	22001
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	3	---	22003
SQL_C_CHAR	4.46.32	SQL_TIME	6	4.46.32	N/A
SQL_C_CHAR	4-46-32	SQL_TIME	6	not applicable	22007
SQL_C_DOUBLE	123.45	SQL_CHAR	22	1.23450000 000000e+02	N/A
SQL_C_FLOAT	1234.56	SQL_FLOAT	not applicable	1234.56	N/A
SQL_C_FLOAT	1234.56	SQL_INTEGER	not applicable	1234	22001
SQL_C_TIMESTAMP	1992-12-31 23:45:55. 123456	SQL_DATE	6	1992-12-31	01004

Note: SQLSTATE 00000 is not returned by SQLGetDiagRec(), rather it is indicated when the function returns SQL_SUCCESS.

Related reference:

- “C data types for CLI applications” on page 43
- “Data conversions supported in CLI” on page 397
- “SQL symbolic and default data types for CLI applications” on page 41
- “SQL to C data conversion in CLI” on page 399

Part 7. Appendixes

Appendix A. DB2 Database technical information

Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
 - Topics
 - Help for DB2 tools
 - Sample programs
 - Tutorials
- DB2 books
 - PDF files (downloadable)
 - PDF files (from the DB2 PDF CD)
 - printed books
- Command line help
 - Command help
 - Message help
- Sample programs

IBM periodically makes documentation updates available. If you access the online version on the DB2 Information Center at ibm.com[®], you do not need to install documentation updates because this version is kept up-to-date by IBM. If you have installed the DB2 Information Center, it is recommended that you install the documentation updates. Documentation updates allow you to update the information that you installed from the *DB2 Information Center CD* or downloaded from Passport Advantage as new information becomes available.

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and Redbooks[™] online at ibm.com. Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how we can improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

Related concepts:

- “Features of the DB2 Information Center” in *Online DB2 Information Center*
- “Sample files” in *Samples Topics*

Related tasks:

- “Invoking command help from the command line processor” in *Command Reference*
- “Invoking message help from the command line processor” in *Command Reference*
- “Updating the DB2 Information Center installed on your computer or intranet server” on page 421

Related reference:

- “DB2 technical library in PDF format” on page 416

DB2 technical library in PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order.

Although the tables identify books available in print, the books might not be available in your country or region.

The information in these books is fundamental to all DB2 users; you will find this information useful whether you are a programmer, a database administrator, or someone who works with DB2 Connect or other DB2 products.

Table 45. DB2 technical information

Name	Form Number	Available in print
<i>Administration Guide: Implementation</i>	SC10-4221	Yes
<i>Administration Guide: Planning</i>	SC10-4223	Yes
<i>Administrative API Reference</i>	SC10-4231	Yes
<i>Administrative SQL Routines and Views</i>	SC10-4293	No
<i>Call Level Interface Guide and Reference, Volume 1</i>	SC10-4224	Yes
<i>Call Level Interface Guide and Reference, Volume 2</i>	SC10-4225	Yes
<i>Command Reference</i>	SC10-4226	No
<i>Data Movement Utilities Guide and Reference</i>	SC10-4227	Yes
<i>Data Recovery and High Availability Guide and Reference</i>	SC10-4228	Yes
<i>Developing ADO.NET and OLE DB Applications</i>	SC10-4230	Yes
<i>Developing Embedded SQL Applications</i>	SC10-4232	Yes
<i>Developing SQL and External Routines</i>	SC10-4373	Yes

Table 45. DB2 technical information (continued)

Name	Form Number	Available in print
<i>Developing Java Applications</i>	SC10-4233	Yes
<i>Developing Perl and PHP Applications</i>	SC10-4234	No
<i>Getting Started with Database Application Development</i>	SC10-4252	Yes
<i>Getting started with DB2 installation and administration on Linux and Windows</i>	GC10-4247	Yes
<i>Message Reference Volume 1</i>	SC10-4238	No
<i>Message Reference Volume 2</i>	SC10-4239	No
<i>Migration Guide</i>	GC10-4237	Yes
<i>Net Search Extender Administration and User's Guide</i> Note: HTML for this document is not installed from the HTML documentation CD.	SH12-6842	Yes
<i>Performance Guide</i>	SC10-4222	Yes
<i>Query Patroller Administration and User's Guide</i>	GC10-4241	Yes
<i>Quick Beginnings for DB2 Clients</i>	GC10-4242	No
<i>Quick Beginnings for DB2 Servers</i>	GC10-4246	Yes
<i>Spatial Extender and Geodetic Data Management Feature User's Guide and Reference</i>	SC18-9749	Yes
<i>SQL Guide</i>	SC10-4248	Yes
<i>SQL Reference, Volume 1</i>	SC10-4249	Yes
<i>SQL Reference, Volume 2</i>	SC10-4250	Yes
<i>System Monitor Guide and Reference</i>	SC10-4251	Yes
<i>Troubleshooting Guide</i>	GC10-4240	No
<i>Visual Explain Tutorial</i>	SC10-4319	No
<i>What's New</i>	SC10-4253	Yes
<i>XML Extender Administration and Programming</i>	SC18-9750	Yes
<i>XML Guide</i>	SC10-4254	Yes
<i>XQuery Reference</i>	SC18-9796	Yes

Table 46. DB2 Connect-specific technical information

Name	Form Number	Available in print
<i>DB2 Connect User's Guide</i>	SC10-4229	Yes
<i>Quick Beginnings for DB2 Connect Personal Edition</i>	GC10-4244	Yes

Table 46. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print
<i>Quick Beginnings for DB2 Connect Servers</i>	GC10-4243	Yes

Table 47. WebSphere Information Integration technical information

Name	Form Number	Available in print
<i>WebSphere Information Integration: Administration Guide for Federated Systems</i>	SC19-1001	Yes
<i>WebSphere Information Integration: ASNCLP Program Reference for Replication and Event Publishing</i>	SC19-1000	Yes
<i>WebSphere Information Integration: Configuration Guide for Federated Data Sources</i>	SC19-1034	No
<i>WebSphere Information Integration: SQL Replication Guide and Reference</i>	SC19-1002	Yes

Note: The DB2 Release Notes provide additional information specific to your product's release and fix pack level. For more information, see the related links.

Related concepts:

- "Overview of the DB2 technical information" on page 415
- "About the Release Notes" in *Release notes*

Related tasks:

- "Ordering printed DB2 books" on page 418

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* CD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation CD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation CD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at <http://publib.boulder.ibm.com/infocenter/db2help/>.

Procedure:

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at <http://www.ibm.com/shop/publications/order>. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
 - Locate the contact information for your local representative from one of the following Web sites:
 - The IBM directory of world wide contacts at www.ibm.com/planetwide
 - The IBM Publications Web site at <http://www.ibm.com/shop/publications/order>. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
 - When you call, specify that you want to order a DB2 publication.
 - Provide your representative with the titles and form numbers of the books that you want to order.

Related concepts:

- "Overview of the DB2 technical information" on page 415

Related reference:

- "DB2 technical library in PDF format" on page 416

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure:

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, ? 08003 displays help for the 08003 SQL state, and ? 08 displays help for the 08 class code.

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*

Accessing different versions of the DB2 Information Center

For DB2 Version 9 topics, the DB2 Information Center URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>.

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

In addition, URL <http://publib.boulder.ibm.com/infocenter/db2help/> will always take you to the DB2 Information Center for the most recently-released version.

Related tasks:

- “Updating the DB2 Information Center installed on your computer or intranet server” on page 421

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

Procedure:

To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
 - To add a new language to the list, click the **Add...** button.

Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.

- To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

To display topics in your preferred language in the Firefox browser:

1. In Firefox, select the **Tools** —> **Options** —> **Languages** button. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.
 - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
 - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Related concepts:

- “Overview of the DB2 technical information” on page 415

Updating the DB2 Information Center installed on your computer or intranet server

If you have a locally-installed DB2 Information Center, updated topics can be available for download. The 'Last updated' value found at the bottom of most topics indicates the current level for that topic.

To determine if there is an update available for the entire DB2 Information Center, look for the 'Last updated' value on the Information Center home page. Compare the value in your locally installed home page to the latest value which is available on the IBM hosted Information Center home page. If they are the same, you have the latest documentation level and no update is required. If they are not the same, you should update your locally-installed Information Center.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to determine if update packages are available from IBM. If update packages are available, use the Update feature to download the packages. (The Update feature is only available in stand-alone mode.)
3. Stop the stand-alone Information Center, and restart the DB2 Information Center service on your computer.

Procedure:

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center service.
 - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
 - On Linux, enter the following command:
`/etc/init.d/db2icdv9 stop`
2. Start the Information Center in stand-alone mode.
 - On Windows:
 - a. Open a command window.
 - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the C:\Program Files\IBM\DB2 Information Center\Version 9 directory.
 - c. Run the `help_start.bat` file using the fully qualified path for the DB2 Information Center:
`<DB2 Information Center dir>\doc\bin\help_start.bat`
 - On Linux:
 - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9` directory.
 - b. Run the `help_start.sh` file using the fully qualified path for the DB2 Information Center:
`<DB2 Information Center dir>/doc/bin/help_start`

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the Update button (🔄). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click **Install Updates**.
5. After the download and installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center.
 - On Windows, run the `help_end.bat` file using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>\doc\bin\help_end.bat
```
 - On Linux, run the `help_end.sh` file using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>/doc/bin/help_end
```
7. Restart the DB2 Information Center service.
 - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
 - On Linux, enter the following command:

```
/etc/init.d/db2icdv9 start
```

The updated DB2 Information Center displays the new and updated topics.

Related concepts:

- “DB2 Information Center installation options” in *Quick Beginnings for DB2 Servers*

Related tasks:

- “Installing the DB2 Information Center using the DB2 Setup wizard (Linux)” in *Quick Beginnings for DB2 Servers*
- “Installing the DB2 Information Center using the DB2 Setup wizard (Windows)” in *Quick Beginnings for DB2 Servers*

DB2 Visual Explain tutorial

The DB2 Visual Explain tutorial helps you learn about analyzing, optimizing, and tuning SQL statements for better performance. Lessons provide step-by-step instructions.

Before you begin:

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2help/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 Visual Explain tutorial:

To view the tutorial, click on the title.

Visual Explain Tutorial

Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

Related concepts:

- “Visual Explain overview” in *Administration Guide: Implementation*

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation

Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site

Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at <http://www.ibm.com/software/data/db2/udb/support.html>

Related concepts:

- “Introduction to problem determination” in *Troubleshooting Guide*
- “Overview of the DB2 technical information” on page 415

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix B. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

Company, product, or service names identified in the documents of the DB2 Version 9 documentation library may be trademarks or service marks of International Business Machines Corporation or other companies. Information on the trademarks of IBM Corporation in the United States, other countries, or both is located at <http://www.ibm.com/legal/copytrade.shtml>.

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 documentation library:

Microsoft, Windows, Windows NT[®], and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Itanium[®], Pentium[®], and Xeon[®] are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Appendix C. Further notices for the DB2 Call Level Interface Guide and Reference

This book incorporates text which is copyright The X/Open Company Limited. The text was taken by permission from:

X/Open CAE Specification, March 1995,
Data Management: SQL Call Level Interface (CLI)
(ISBN: 1-85912-081-4, C451).

X/Open Preliminary Specification, March 1995,
Data Management: Structured Query Language (SQL), Version 2
(ISBN: 1-85912-093-8, P446).

This book incorporates text which is copyright 1992, 1993, 1994, 1997 by Microsoft Corporation. The text was taken by permission from Microsoft's *ODBC 2.0 Programmer's Reference and SDK Guide* ISBN 1-55615-658-8, and from Microsoft's *ODBC 3.0 Software Development Kit and Programmer's Reference* ISBN 1-57231-516-4.

Index

Special characters

- (percent)
 - in input to catalog functions 168
 - in LIKE predicates 168
- (underscore)
 - in input to catalog functions 168
 - in LIKE predicates 168

A

- ABS scalar function 174
- absolute value scalar function 174
- ACOS scalar function
 - in vendor escape clause 174
- allocating CLI handles
 - transaction processing 22
- AllowGetDataColumnReaccess CLI/ODBC configuration keyword 284
- AltHostName CLI/ODBC keyword 284
- AltPort CLI/ODBC keyword 285
- APD descriptor 151
- AppendAPIName CLI/ODBC configuration keyword 285
- AppendForFetchOnly CLI/ODBC configuration keyword 287
- AppendRowColToErrorMessage CLI/ODBC configuration keyword 286
- application parameter descriptor (APD) 151
- application row descriptor (ARD) 151
- AppUsesLobLocator CLI/ODBC configuration keyword 287
- ARD descriptor 151
- array input
 - column-wise 81
 - row-wise 82
- array output 87
- ArrayInputChain CLI/ODBC configuration keyword 288
- ASCII scalar function
 - for vendor escape clauses 174
- ASIN scalar function
 - for vendor escape clauses 174
- AsyncEnable CLI/ODBC configuration keyword 289
- asynchronous
 - executing CLI functions 129
 - overview 127
- ATAN scalar function
 - in vendor escape clauses 174
- ATAN2 scalar function
 - in vendor escape clauses 174
- attributes
 - connection 163
 - environment 163
 - querying and setting 163
 - statement 163
- Authentication CLI/ODBC keyword 290
- AutoCommit CLI/ODBC configuration keyword 290

B

- BIDI CLI/ODBC keyword 291
- BIGINT SQL data type
 - conversion to C 399

- BINARY SQL data type
 - conversion to C 399
- bind files
 - and package names 203
- binding
 - and limitations for CLI packages 205
 - application variables 26, 91
 - columns 91
 - columns in CLI 85
 - parameter markers 26
 - column-wise 81
 - row-wise 82
- BitData CLI/ODBC configuration keyword 292
- bldapp
 - AIX 224
 - HP-UX 227
 - Linux 227
 - Solaris 228
 - Windows 233
- bldrtn
 - AIX 225
 - HP-UX 227
 - Linux 227
 - Solaris 228
 - Windows 233
- BLOB SQL data type
 - conversion to C 399
- BLOBs (binary large objects)
 - CLI applications 96
- BlockForNRows CLI/ODBC configuration keyword 292
- BlockLobs CLI/ODBC configuration keyword 293
- bookmarks in CLI
 - deleting bulk data with 109
 - description 78
 - inserting bulk data with 106
 - result set terminology 70
- build script
 - AIX applications 224
 - AIX routines 225
 - HP-UX applications 227
 - HP-UX routines 227
 - Linux applications 227
 - Linux routines 227
 - Solaris applications 228
 - Solaris routines 228
 - Windows applications 233
 - Windows routines 233
- building CLI applications
 - UNIX 219
 - multi-connection 220
 - Windows 228
 - multi-connection 229
 - with configuration files 224
- building CLI routines
 - UNIX 222
 - Windows 231
 - with configuration files 226
- bulk data
 - deleting, in CLI 109
 - inserting, in CLI 106

C

C

- data types 43
- call level interface (CLI)
 - compared with embedded SQL 4
- capture file 187
- case sensitivity
 - cursor name arguments 47
- catalog functions 167
- catalogs
 - querying 167
- CEILING scalar function 174
- CHAR scalar function
 - for CLI applications 174
- CHAR SQL data type
 - conversion to C 399
- character strings
 - interpreting 47
 - length 47
- CheckForFork CLI/ODBC configuration keyword 294
- CICS (Customer Information Control System)
 - running applications on 141
- CLI
 - drivers 7
 - registering XA library with DTC 252
 - IBM DB2 Driver for ODBC and CLI
 - configuring 249, 250, 253, 254
 - connecting to databases 257
 - DB2 registry variables 251
 - deploying with applications 269
 - environment variables 251
 - installing 245, 246, 247
 - LDAP support 265
 - license requirements 270
 - obtaining 243
 - overview 241
 - problem determination and investigation 266
 - restrictions 266
 - running database applications 263
 - supported CLI and ODBC functions 264
 - supported XA functions 265
 - uninstalling 273
- CLI (call level interface)
 - applications
 - issuing SQL 23
 - terminating 53
 - array input chaining 62
 - binding parameter markers 29
 - bookmarks
 - deleting bulk data 109
 - inserting bulk data 106
 - retrieving bulk data 105
 - retrieving data 79
 - updating bulk data 107
 - building applications
 - multi-connection, UNIX 220
 - multi-connection, Windows 229
 - UNIX 219
 - Windows 228
 - with configuration files 224
 - building routines
 - UNIX 222
 - Windows 231
 - with configuration files 226
 - bulk data
 - deleting 109
 - inserting 106
 - CLI (call level interface) (*continued*)
 - bulk data (*continued*)
 - retrieving 105
 - updating 107
 - compound SQL 123
 - return codes 125
 - configuration keywords 279
 - cursors 65
 - selection 68
 - deferred prepare 26
 - deleting data 35
 - descriptors 151
 - consistency checks 154
 - diagnostics overview 49
 - environmental setup 209
 - executing SQL 24
 - functions
 - Unicode 144
 - handles
 - description 15
 - freeing 38
 - initializing 18
 - introduction 3
 - issuing SQL 23
 - keywords 279
 - LOB locators 98
 - long data 103
 - multithreaded applications
 - mixed 136
 - model 134
 - performance improvement
 - array input chaining 62
 - preparing SQL 24
 - retrieving array data
 - column-wise binding 89
 - row-wise binding 90
 - retrieving data with bookmarks 79
 - retrieving query results 33
 - sample program files 235
 - location of 235
 - SQL/XML functions 117
 - static profiling 185
 - stored procedures
 - calling 113
 - commit behavior 115
 - trace facility 189
 - trace files 194
 - Unicode
 - applications 143
 - functions 144
 - ODBC driver managers 145
 - updating data 35
 - XML data 117
 - changing default type 120
 - inserts 118
 - retrieval 119
 - updates 118
 - XQuery expressions 117
 - CLI applications
 - XML data 117
 - CLI functions
 - executing asynchronously 129
 - overview of asynchronous execution 127
 - CLI/ODBC keywords
 - AllowGetDataColumnReaccess 284
 - AltHostName 284
 - AltPort 285

CLI/ODBC keywords (continued)

AppendAPIName 285
 AppendForFetchOnly 287
 AppendRowColToErrorMessage 286
 AppUsesLobLocator 287
 ArrayInputChain 288
 AsyncEnable 289
 Authentication 290
 AutoCommit 290
 BIDI 291
 BitData 292
 BlockForNRows 292
 BlockLobs 293
 CheckForFork 294
 ClientAcctStr 294
 ClientApplName 295
 ClientBuffersUnboundLOBS 296
 ClientUserID 297
 ClientWrkStnName 298
 CLIPkg 299
 ConnectNode 299
 ConnectTimeout 300
 ConnectType 301
 CurrentFunctionPath 302
 CurrentImplicitXMLParseOption 302
 CurrentMaintainedTableTypesForOpt 303
 CurrentPackagePath 304
 CurrentPackageSet 305
 CurrentRefreshAge 306
 CurrentSchema 306
 CurrentSQLID 306
 CursorHold 307
 CursorTypes 308
 Database 309
 DateTimeStringFormat 310
 DB2Degree 311
 DB2Explain 311
 DB2NETNamedParam 312
 DB2Optimization 313
 DBAlias 313
 DBName 314
 DecimalFloatRoundingMode 315
 DeferredPrepare 316
 DescribeCall 317
 DescribeInputOnPrepare 318
 DescribeOutputLevel 318
 DescribeParam 320
 DiagLevel 321
 DiagPath 321
 DisableKeysetCursor 321
 DisableMultiThread 322
 DisableUnicode 322
 DSN 323
 FileDSN 323
 FloatPrecRadix 324
 GranteeList 324
 GrantorList 325
 Graphic 326
 Hostname 327
 IgnoreWarnings 327
 IgnoreWarnList 328
 initialization file 277
 Instance 328
 Interrupt 329
 KeepDynamic 329
 KRBPlugin 330
 listing by category 279

CLI/ODBC keywords (continued)

LoadXAInterceptor 330
 LOBCacheSize 331
 LOBFileThreshold 332
 LOBMaxColumnSize 332
 LockTimeout 333
 LongDataCompat 333
 MapBigintCDefault 334
 MapCharToWChar 335
 MapDateCDefault 336
 MapDateDescribe 337
 MapDecimalFloatDescribe 338
 MapGraphicDescribe 339
 MapTimeCDefault 340
 MapTimeDescribe 340
 MapTimestampCDefault 342
 MapTimestampDescribe 342
 MapXMLCDefault 120, 343
 MapXMLDescribe 120, 344
 MaxLOBBlockSize CLI/ODBC keyword 345
 Mode 346
 NotifyLevel 346
 OleDbReportIsLongForLongTypes 347
 OleDbReturnCharAsWChar 348
 OleDbSQLColumnsSortByOrdinal 348
 OptimizeForNRows 349
 Patch1 350
 Patch2 352
 Port 355
 ProgramName 356
 Protocol 356
 PWD 357
 PWDPlugin 358
 QueryTimeoutInterval 358
 ReadCommonSectionOnNullConnect 359
 ReceiveTimeout 360
 Reopt 360
 ReportPublicPrivileges 362
 ReportRetryErrorsAsWarnings 361
 RetCatalogAsCurrServer 362
 RetOleDbConnStr 363
 RetryOnError 364
 SaveFile 365
 SchemaList 365
 ServerMsgMask 366
 ServiceName 366
 SkipTrace 367
 SQLOverrideFileName 368
 StaticCapFile 368
 StaticLogFile 369
 StaticMode 370
 StaticPackage 370
 StreamGetData 371
 StreamPutData 372
 SysSchema 372
 TableType 374
 TempDir 374
 Trace 375
 TraceComm 376
 TraceErrImmediate 377
 TraceFileName 378
 TraceFlush 379
 TraceFlushOnError 379
 TraceLocks 380
 TracePathName 381
 TracePIDList 382
 TracePIDTID 383

CLI/ODBC keywords (*continued*)
 TraceRefreshInterval 383
 TraceStmtOnly 384
 TraceTime 385
 TraceTimestamp 386
 Trusted_Connection 386
 TxnIsolation 388
 UID 388
 Underscore 389
 UseOldStpCall 390
 UseServerMsgSP 291
 WarningList 391
 XMLDeclaration 392
 CLI/ODBC/JDBC
 static profiling
 capture file 187
 creating static SQL 185
 trace
 facility 189
 files 194
 ClientAcctStr CLI/ODBC configuration keyword 294
 ClientAppName CLI/ODBC configuration keyword 295
 ClientBuffersUnboundLOBS CLI/ODBC configuration keyword 296
 ClientUserID CLI/ODBC configuration keyword 297
 ClientWrkStnName CLI/ODBC configuration keyword 298
 CLIPkg CLI/ODBC configuration keyword 299
 CLOB (character large object)
 data type
 CLI applications 96
 conversion to C 399
 column binding offsets 91
 column-wise binding 89
 columns
 binding, in CLI 85
 commit
 behavior
 in CLI stored procedures 115
 transactions 30
 compound SQL
 CLI
 executing in 123
 return codes 125
 CONCAT scalar function
 CLI 174
 concise descriptor functions 160
 connections
 attributes
 changing 163
 connection strings 163
 multiple 55
 ConnectNode CLI/ODBC configuration keyword 299
 ConnectTimeout CLI/ODBC configuration keyword 300
 ConnectType CLI/ODBC configuration keyword 301
 contacting IBM 433
 conversions
 data types, in CLI 397
 CONVERT scalar function 174
 coordinated transactions
 distributed 137
 establishing 138
 copying
 descriptors
 in CLI applications 158
 core level functions 3
 COS scalar function
 for CLI applications 174

 COT scalar function
 for CLI applications 174
 CURDATE scalar function 174
 CurrentFunctionPath CLI/ODBC configuration keyword 302
 CurrentImplicitXMLParseOption CLI/ODBC configuration keyword 302
 CurrentMaintainedTableTypesForOpt CLI/ODBC configuration keyword 303
 CurrentPackagePath CLI/ODBC configuration keyword 304
 CurrentPackageSet CLI/ODBC configuration keyword 305
 CurrentRefreshAge CLI/ODBC configuration keyword 306
 CurrentSchema CLI/ODBC configuration keyword 306
 CurrentSQLID CLI/ODBC configuration keyword 306
 CursorHold CLI/ODBC configuration keyword 307
 cursors
 CLI (call level interface)
 bookmarks 78
 considerations 65
 selection 68
 dynamic scrollable 65
 holding across rollbacks 55
 scrollable
 retrieving data with in CLI 76
 CursorTypes CLI/ODBC configuration keyword 308
 CURTIME scalar function 174

D
 data
 conversions
 CLI 397
 data conversion
 C data types 41
 C to SQL data types 406
 data types 39
 default data types 41
 description 39
 SQL data types 41
 SQL to C data types 399
 data retrieval
 in pieces, CLI 95
 data types
 C language 41
 C, in CLI 43
 conversion
 CLI 397
 SQL 41
 Database CLI/ODBC configuration keyword 309
 DATABASE scalar function 174
 DATE SQL data type
 conversion to C 399
 DateTimeStringFormat CLI/ODBC configuration keyword 310
 DAYNAME scalar function
 for CLI applications 174
 DAYOFMONTH scalar function 174
 DAYOFWEEK scalar function
 for CLI applications 174
 DAYOFWEEK_ISO scalar function
 for CLI applications 174
 DAYOFYEAR scalar function 174
 DB2
 as transaction manager 138
 DB2 CLI
 sample program files 235
 DB2 copies
 CLI/ODBC applications 217

DB2 Information Center
 updating 421
 versions 419
 viewing in different languages 420
 DB2_ENABLE_LDAP
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2_FORCE_NLS_CACHE
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2_NO_FORK_CHECK
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2ACCOUNT
 as environment variable
 with DB2 IBM DB2 Driver for ODBC and CLI 251
 DB2BIDI
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 db2cli.ini file
 attributes 163
 description 277
 DB2CLIINIPATH
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2CODEPAGE
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2COUNTRY
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2Degree CLI/ODBC configuration keyword 311
 DB2DOMAINLIST
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2Explain CLI/ODBC configuration keyword 311
 DB2GRAPHICUNICODESERVER
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2LDAP_BASEDN
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2LDAP_CLIENT_PROVIDER
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2LDAP_KEEP_CONNECTION
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2LDAP_SEARCH_SCOPE
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2LDAPHOST
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2LOCALE
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2NETNamedParam CLI/ODBC configuration
 keyword 312
 DB2NODE 299
 DB2NOEXITLIST
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2Optimization CLI/ODBC configuration keyword 313
 db2oregl.exe 252
 DB2SORCVBUF
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2SOSNDBUF
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2TCP_CLIENT_RCVMTIMEOUT
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DB2TERRITORY
 as environment variable
 with IBM DB2 Driver for ODBC and CLI 251
 DBAlias CLI/ODBC configuration keyword 313
 DBCLOB data type
 description 96
 DBCLOB SQL data type
 conversion to C 399
 DBName CLI/ODBC configuration keyword 314
 DECIMAL data type
 conversion to C 399
 DecimalFloatRoundingMode CLI/ODBC configuration
 keyword 315
 deferred arguments 26
 deferred prepare
 in CLI applications 26
 DeferredPrepare CLI/ODBC configuration keyword 316
 DEGREES scalar function
 for CLI applications 174
 deleting
 bulk data in CLI 109
 data in CLI 35
 DescribeCall CLI/ODBC configuration keyword 317
 DescribeInputOnPrepare CLI/ODBC configuration
 keyword 318
 DescribeOutputLevel CLI/ODBC configuration keyword 318
 DescribeParam CLI/ODBC configuration keyword 320
 descriptor handles 151
 descriptors 151
 allocating 155
 concise functions 160
 consistency checks 154
 copying 158
 freeing 155
 header fields 151
 records 151
 types 151
 DiagLevel CLI/ODBC keyword 321
 diagnostics 49
 DiagPath CLI/ODBC keyword 321
 DIFFERENCE scalar function
 extended scalar function in CLI applications 174
 DisableKeysetCursor CLI/ODBC configuration keyword 321
 DisableMultiThread CLI/ODBC configuration keyword 322
 DisableUnicode CLI/ODBC configuration keyword 322
 distinct types
 description 147
 distributed transactions 137
 distributed unit of work 137
 CICS 141
 DB2 as transaction manager 138
 description 137
 Encina 141
 processor based transaction manager 141
 documentation 415, 416
 terms and conditions of use 423
 DOUBLE data type
 conversion to C 399

- drivers
 - CLI 7, 9
 - manager 9
 - ODBC 7, 9
- DSN CLI/ODBC keyword 323
- DTC
 - registering XA library with db2oregl.exe 252

E

- embedded SQL
 - compared with CLI 4
 - mixing with DB2 CLI 183
- Encina, running applications on 141
- environment attributes
 - changing 163
- ESCAPE clauses
 - vendor 171
- examples
 - distinct types
 - CLI applications 147
- executing
 - SQL in CLI 24
- EXP scalar function 174

F

- fetching
 - LOB data in CLI 99
- File DSN
 - database to connect 309
 - host name 327
 - IP address 327
 - protocol used 356
 - service name 366
- file input/output for LOB data in CLI 101
- FileDSN CLI/ODBC keyword 323
- FLOAT SQL data type
 - conversion to C 399
- FloatPrecRadix CLI/ODBC configuration keyword 324
- FLOOR scalar function 174
- freeing CLI handles
 - in CLI application 38
- freeing statement resources in CLI 37

G

- GranteeList CLI/ODBC configuration keyword 324
- GrantorList CLI/ODBC configuration keyword 325
- Graphic CLI/ODBC configuration keyword 326
- GRAPHIC SQL data type
 - conversion to C 399

H

- handles
 - descriptor 151
 - freeing 38
 - types 15
- help
 - displaying 420
 - for SQL statements 419
- Hostname CLI/ODBC configuration keyword 327
- HOURLY scalar function 174

I

- IBM DB2 Driver for ODBC and CLI
 - CLI trace 266
 - configuring 249
 - environment variables 250
 - Microsoft DTC 253
 - Microsoft ODBC driver manager 254
 - registering ODBC data sources 259
 - connecting to databases 257
 - DB2 registry variables
 - supported as environment variables 251
 - DB2 trace 266
 - db2diag.log 266
 - db2support 266
 - deploying with applications 269
 - environment variables 251
 - installing 245
 - multiple copies 246
 - with an existing client 247
 - LDAP support 265
 - license requirements 270
 - obtaining 243
 - overview 241
 - problem determination and investigation 266
 - registering ODBC data sources 259
 - restrictions 266
 - running applications 263
 - supported DB2 CLI and ODBC functions 264
 - supported XA functions 265
 - uninstalling 273
- IFNULL scalar function 174
- IgnoreWarnings CLI/ODBC configuration keyword 327
- IgnoreWarnList CLI/ODBC configuration keyword 328
- implementation parameter descriptor (IPD) 151
- implementation row descriptor (IRD) 151
- importing
 - data
 - with the CLI LOAD utility 110
- IN DATABASE statement 314
- Information Center
 - updating 421
 - versions 419
 - viewing in different languages 420
- INI file
 - db2cli.ini 277
- initialization
 - task 17
- initialization file
 - purpose 163
- initializing
 - CLI applications 18
- INSERT scalar function 174
- inserting data
 - XML
 - CLI applications 118
- Instance CLI/ODBC keyword 328
- INTEGER SQL data type
 - conversion to C 399
- Interrupt CLI/ODBC keyword 329
- INVALID_HANDLE 50
- IPD descriptor 151
- IRD descriptor 151
- isolation levels
 - ODBC 9
- issuing SQL in CLI 23

J

JULIAN_DAY scalar function 174

K

KeepDynamic CLI/ODBC configuration keyword 329
keysets 70
KRBPlugin CLI/ODBC keyword 330

L

large object (LOB) data types
 fetching with locators in CLI 99
 file input and output in CLI 101
 in CLI applications 96
 in ODBC applications 102
 LongDataCompat CLI/ODBC configuration keyword 102
LCASE scalar function
 description 174
LDAP (Lightweight Directory Access Protocol)
 IBM DB2 Driver for ODBC and CLI 265
LEFT scalar function
 description 174
LENGTH scalar function 174
license policies
 IBM DB2 Driver for ODBC and CLI 270
load utility
 callable from CLI 110
LoadXAInterceptor CLI/ODBC configuration keyword 330
LOB (large object) data types
 fetching with locators in CLI 99
 file input and output in CLI 101
 in CLI applications 96
 in ODBC applications 102
 LongDataCompat CLI/ODBC keyword 102
LOB locators 98, 99
LOBCacheSize CLI/ODBC configuration keyword 331
LOBFileThreshold CLI/ODBC configuration keyword 332
LOBMaxColumnSize CLI/ODBC configuration keyword 332
LOCATE scalar function
 listed 174
LockTimeout CLI/ODBC configuration keyword 333
LOG scalar function 174
LOG10 scalar function 174
long data
 inserts and updates, in CLI 103
 retrieving data in pieces 93
 sending data in pieces 93
LongDataCompat CLI/ODBC configuration keyword 102,
 333
LONGVARBINARY data type
 conversion to C 399
LONGVARCHAR data type
 conversion to C 399
LONGVARGRAPHIC data type
 conversion to C 399
lowercase conversion scalar function 174
LTRIM scalar function
 listed 174

M

MapBigintCDefault CLI/ODBC configuration keyword 334
MapCharToWChar CLI/ODBC configuration keyword 335
MapDateCDefault CLI/ODBC configuration keyword 336

MapDateDescribe CLI/ODBC configuration keyword 337
MapDecimalFloatDescribe CLI/ODBC configuration
 keyword 338
MapGraphicDescribe CLI/ODBC configuration keyword 339
MapTimeCDefault CLI/ODBC configuration keyword 340
MapTimeDescribe CLI/ODBC configuration keyword 340
MapTimestampCDefault CLI/ODBC configuration
 keyword 342
MapTimestampDescribe CLI/ODBC configuration
 keyword 342
MapXMLCDefault CLI/ODBC configuration keyword 343
MapXMLDescribe CLI/ODBC configuration keyword 344
MaxLOBBlockSize CLI/ODBC configuration keyword 345
metadata
 characters 168
Microsoft DTC
 configuring
 IBM DB2 Driver for ODBC and CLI 253
Microsoft ODBC 9
Microsoft ODBC driver manager
 configuring
 IBM DB2 Driver for ODBC and CLI 254
midnight, seconds since scalar function 174
MINUTE scalar function
 for CLI applications 174
mixing DB2 APIs and DB2 CLI
 multithreaded 136
mixing embedded SQL and DB2 CLI 183
 multithreaded 136
MOD scalar function 174
Mode CLI/ODBC configuration keyword 346
MONTH scalar function 174
MONTHNAME scalar function 174
multi-threaded applications 133
multi-threaded applications, CLI
 model 134
multisite updates
 CLI applications 137

N

native error code 51
notices 425
NotifyLevel CLI/ODBC keyword 346
NOW scalar function 174
null-terminated strings
 in CLI applications 47
NUMERIC SQL data type
 conversion to C 399

O

ODBC
 driver managers
 unixODBC 212, 213
 drivers 7
 registering XA library with DTC 252
 IBM DB2 Driver for ODBC and CLI
 configuring 249, 250, 253, 254
 connecting to databases 257
 DB2 registry variables 251
 deploying with applications 269
 environment variables 251
 installing 245, 246, 247
 LDAP support 265
 license requirements 270

- ODBC (*continued*)
 - IBM DB2 Driver for ODBC and CLI (*continued*)
 - obtaining 243
 - overview 241
 - problem determination and investigation 266
 - registering ODBC data sources 259
 - restrictions 266
 - running database applications 263
 - supported CLI and ODBC functions 264
 - supported XA functions 265
 - uninstalling 273
 - registering ODBC data sources
 - IBM DB2 Driver for ODBC and CLI 259
- ODBC (open database connectivity)
 - and DB2 CLI 3, 9
 - core level functions 3
 - isolation levels 9
 - setting up UNIX environment 210
 - vendor escape clauses 171
- offsets
 - binding columns 91
 - changing parameter bindings 84
- OleDbReportIsLongForLongTypes CLI/ODBC configuration keyword 347
- OleDbReturnCharAsWChar CLI/ODBC configuration keyword 348
- OleDbSQLColumnsSortByOrdinal CLI/ODBC configuration keyword 348
- OptimizeForNRows CLI/ODBC configuration keyword 349
- options
 - connection 163
 - environment 163
 - querying and setting 163
 - statement 163
- ordering DB2 books 418

P

- package names
 - and binding 203
- packages
 - and bind option limitations 205
- parallelism
 - setting degree of 311
- parameter markers
 - binding
 - column-wise array input, in CLI 81
 - in CLI applications 26, 29
 - row-wise array input, in CLI 82
 - changing bindings 84
- parameter status array, CLI 83
- parameters
 - diagnostics, in CLI 83
- parsing
 - explicit
 - CLI applications 118
 - implicit
 - CLI applications 118
- Patch1 CLI/ODBC configuration keyword 350
- Patch2 CLI/ODBC configuration keyword 352
- pattern values 168
- performance
 - CLI array input chaining 62
- PI scalar function 174
- Port CLI/ODBC configuration keyword 355
- POWER scalar function
 - list 174

- prepared SQL statements
 - in CLI applications
 - creating 24
- printed books
 - ordering 418
- problem determination
 - online information 423
 - tutorials 423
- process-based transaction manager 141
- ProgramName CLI/ODBC configuration keyword 356
- Protocol CLI/ODBC configuration keyword 356
- PWD CLI/ODBC configuration keyword 357
- PWDPlugin CLI/ODBC keyword 358

Q

- QUARTER scalar function 174
- queries
 - system catalog information 167
- QueryTimeoutInterval CLI/ODBC configuration keyword 358

R

- RADIANS scalar function
 - in CLI applications 174
- RAND scalar function
 - in CLI applications 174
- ReadCommonSectionOnNullConnect CLI/ODBC configuration keyword 359
- REAL SQL data type
 - conversion
 - to C data type 399
- ReceiveTimeout CLI/ODBC configuration keyword 360
- reentrance 133
- Reopt CLI/ODBC configuration keyword 360
- REPEAT scalar function
 - list 174
- REPLACE scalar function
 - list 174
- ReportPublicPrivileges CLI/ODBC configuration keyword 362
- ReportRetryErrorsAsWarnings CLI/ODBC configuration keyword 361
- result sets
 - specifying rowset returned from, in CLI 73
 - terminology, CLI 70
- RetCatalogAsCurrServer CLI/ODBC configuration keyword 362
- RetOleDbConnStr CLI/ODBC configuration keyword 363
- retrieving data
 - array
 - column-wise binding 89
 - row-wise binding 90
 - bulk, with bookmarks in CLI 105
 - CLI 87
 - in pieces, CLI 95
 - with bookmarks in CLI 79
 - with scrollable cursors, in CLI 76
- XML
 - CLI applications 119
- retrieving query results
 - CLI 33
- retrieving row sets
 - CLI examples 71
- RetryOnError CLI/ODBC configuration keyword 364

- return codes
 - CLI
 - compound SQL 125
 - functions 50
- RIGHT scalar function
 - vendor escape clauses 174
- rollback
 - transactions 30
- ROUND scalar function
 - vendor escape clauses 174
- row sets
 - description 70
 - retrieval examples, in CLI 71
 - specifying, in CLI 73
- row-wise binding 87, 90
- RTRIM scalar function
 - vendor escape clauses 174

S

- samples
 - programs
 - CLI, location of 235
- SaveFile CLI/ODBC keyword 365
- SchemaList CLI/ODBC configuration keyword 365
- search conditions
 - in input to catalog functions 168
- SECOND scalar function
 - in CLI applications 174
- SECONDS_SINCE_MIDNIGHT scalar function 174
- serialization
 - explicit
 - CLI applications 119
 - implicit
 - CLI applications 117, 119
- ServerMsgMask CLI/ODBC configuration keyword 366
- ServiceName CLI/ODBC configuration keyword 366
- SET CURRENT SCHEMA statement 306
- settings
 - CLI environment
 - run-time support 209
 - Windows 215
- SIGN scalar function
 - list 174
- SIN scalar function
 - list 174
- SkipTrace CLI/ODBC configuration keyword 367
- SMALLINT data type
 - conversion to C/C++ 399
- SOUNDEX scalar function
 - in CLI applications 174
- SPACE scalar function
 - list 174
- SQL (Structured Query Language)
 - parameter markers 26
- SQL Access Group 3
- SQL statements
 - displaying help 419
- SQL_ATTR_
 - APP_USES_LOB_LOCATOR
 - AppUsesLOBLocator 287
 - APPEND_FOR_FETCH_ONLY 287
 - ASYNC_ENABLE
 - AsyncEnable 289
 - AUTOCOMMIT
 - AutoCommit 290

- SQL_ATTR_ (continued)
 - BLOCK_LOBS
 - BlockLobs 293
 - CONNECT_NODE
 - ConnectNode 299
 - CONNECTTYPE 138
 - ConnectType 301
 - CURRENT_PACKAGE_PATH
 - CurrentPackagePath 304
 - CURRENT_PACKAGE_SET
 - CurrentPackageSet 305
 - CURSOR_HOLD
 - CursorHold 307
 - DB2EXPLAIN
 - DB2Explain 311
 - DECFLOAT_ROUNDING_MODE
 - DecimalFloatRoundingMode 315
 - DEFERRED_PREPARE
 - DeferredPrepare 316
 - DESCRIBE_CALL
 - DescribeCall 317
 - DESCRIBE_OUTPUT_LEVEL 318
 - INFO_ACCTSTR
 - ClientAcctStr 294
 - INFO_APPLNAME
 - ClientAppName 295
 - INFO_PROGRAMNAME
 - ProgramName 356
 - INFO_USERID
 - ClientUserID 297
 - INFO_WRKSTNNAME
 - ClientWrkStnName 298
 - KEEP_DYNAMIC
 - KeepDynamic 329
 - LOB_CACHE_SIZE
 - LOBCacheSize 331
 - LOGIN_TIMEOUT
 - ConnectTimeout 300
 - LONGDATA_COMPAT 102
 - LongDataCompat 333
 - MAPCHAR
 - MapCharToWChar 335
 - MAX_LOB_BLOCK_SIZE
 - MaxLOBBlockSize 345
 - OPTIMIZE_FOR_NROWS
 - OptimizeForNRows 349
 - PROCESSCTRL
 - CheckForFork 294
 - QUERY_TIMEOUT
 - QueryTimeoutInterval 358
 - RECEIVE_TIMEOUT
 - ReceiveTimeout 360
 - REOPT
 - Reopt 360
 - REPORT_ISLONG_FOR_LONGTYPES_OLEDB
 - OleDbReportIsLongForLongTypes 347
 - SERVER_MSGTXT_MASK
 - ServerMsgMask 366
 - SERVER_MSGTXT_SP
 - UseServerMsgSP 391
 - SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB
 - OleDbSQLColumnsSortByOrdinal 348
 - STREAM_GETDATA
 - StreamGetData 371
 - TRACE
 - Trace 375

SQL_ATTR_ (continued)
 TXN_ISOLATION
 TxnIsolation 388

SQL_C_BINARY 406

SQL_C_BIT 406

SQL_C_CHAR 406

SQL_C_DATE 406

SQL_C_DBCHAR 406

SQL_C_DOUBLE 406

SQL_C_FLOAT 406

SQL_C_LONG 406

SQL_C_SHORT 406

SQL_C_TIME 406

SQL_C_TIMESTAMP 406

SQL_C_TINYINT 406

SQL_CONCURRENT_TRANS 138

SQL_COORDINATED_TRANS 138

SQL_ERROR 50

SQL_NEED_DATA 50

SQL_NO_DATA_FOUND 50

SQL_NTS 47

SQL_ONEPHASE 138

SQL_STILL_EXECUTING 50

SQL_SUCCESS 50

SQL_SUCCESS_WITH_INFO 50

SQL_TWOPHASE 138

SQLAllocStmt deprecated CLI function 20

SQLBindCol CLI function 20

SQLBindParameter function 26

SQLBrowseConnect CLI function
 Unicode version 144

SQLBrowseConnectW CLI function 144

SQLBulkOperations CLI function
 deleting bulk data 109
 inserting bulk data 106
 retrieving bulk data 105
 updating bulk data 107

SQLColAttribute CLI function
 Unicode version 144

SQLColAttributes CLI function
 overview 20
 Unicode version 144

SQLColAttributesW CLI function 144

SQLColAttributeW CLI function 144

SQLColumnPrivileges CLI function
 Unicode version 144

SQLColumnPrivilegesW CLI function 144

SQLColumns CLI function
 Unicode version 144

SQLColumnsW CLI function 144

SQLConnect CLI function
 Unicode version 144

SQLConnectW CLI function 144

SQLDataSources CLI function
 overview 20
 Unicode version 144

SQLDataSourcesW CLI function 144

SQLDescribeCol CLI function
 overview 20
 Unicode version 144

SQLDescribeColW CLI function 144

SQLDriverConnect CLI function
 default values 163
 Trusted_connection CLI/ODBC configuration
 keyword 386
 Unicode version 144

SQLDriverConnectW CLI function 144

SQLEndTran CLI function 32

SQLError CLI function 144

SQLErrorW CLI function 144

SQLExecDirect CLI function
 overview 20
 Unicode version 144

SQLExecDirectW CLI function 144

SQLExecute CLI function
 overview 20

SQLExtendedPrepare CLI function
 Unicode version 144

SQLExtendedPrepareW CLI function 144

SQLFetch CLI function
 overview 20

SQLForeignKeys CLI function
 Unicode version 144

SQLForeignKeysW CLI function 144

SQLFreeStmt CLI function
 overview 20

SQLGetConnectAttr CLI function
 Unicode version 144

SQLGetConnectAttrW CLI function 144

SQLGetConnectOption CLI function 144

SQLGetConnectOptionW CLI function 144

SQLGetCursorName CLI function
 Unicode version 144

SQLGetCursorNameW CLI function 144

SQLGetData CLI function
 overview 20

SQLGetDescField CLI function
 Unicode version 144

SQLGetDescFieldW CLI function 144

SQLGetDescRec CLI function
 Unicode version 144

SQLGetDescRecW CLI function 144

SQLGetDiagField CLI function
 Unicode version 144

SQLGetDiagFieldW CLI function 144

SQLGetDiagRec CLI function
 Unicode version 144

SQLGetDiagRecW CLI function 144

SQLGetInfo CLI function
 Unicode version 144

SQLGetInfoW CLI function 144

SQLGetStmtAttr CLI function
 Unicode version 144

SQLGetStmtAttrW CLI function 144

SQLNativeSql CLI function
 Unicode version 144

SQLNativeSqlW CLI function 144

SQLNumResultCols CLI function
 overview 20

SQLOverrideFileName CLI/ODBC configuration
 keyword 368

SQLPrepare CLI function
 overview 20
 Unicode version 144

SQLPrepareW CLI function 144

SQLPrimaryKeys CLI function
 Unicode version 144

SQLPrimaryKeysW CLI function 144

SQLProcedureColumns CLI function
 Unicode version 144

SQLProcedureColumnsW CLI function 144

SQLProcedures CLI function
 Unicode version 144

SQLProceduresW CLI function 144

- SQLRowCount CLI function
 - overview 20
- SQLSetConnectAttr CLI function
 - Unicode version 144
- SQLSetConnectAttrW CLI function 144
- SQLSetConnectOption deprecated CLI function
 - Unicode version 144
- SQLSetConnectOptionW CLI function 144
- SQLSetCursorName CLI function
 - Unicode version 144
- SQLSetCursorNameW CLI function 144
- SQLSetDescField CLI function
 - Unicode version 144
- SQLSetDescFieldW CLI function 144
- SQLSetParam deprecated CLI function 20
- SQLSetStmntAttr CLI function
 - Unicode version 144
- SQLSetStmntAttrW CLI function 144
- SQLSpecialColumns CLI function
 - Unicode version 144
- SQLSpecialColumnsW CLI function 144
- SQLSTATE
 - format 51
- SQLStatistics CLI function
 - Unicode version 144
- SQLStatisticsW CLI function 144
- SQLTablePrivileges CLI function
 - Unicode version 144
- SQLTablePrivilegesW CLI function 144
- SQLTables CLI function
 - Unicode version 144
- SQLTablesW CLI function 144
- SQRT scalar function
 - list 174
- statement attributes
 - changing 163
- statement handles
 - allocating 22
- statements
 - freeing resources, in CLI 37
- StaticCapFile CLI/ODBC configuration keyword 368
- StaticLogFile CLI/ODBC configuration keyword 369
- StaticMode CLI/ODBC configuration keyword 370
- StaticPackage CLI/ODBC configuration keyword 370
- stored procedures
 - calling
 - CLI applications 113
 - ODBC escape clause 171
- StreamGetData CLI/ODBC configuration keyword 371
- StreamPutData CLI/ODBC configuration keyword 372
- strings
 - input arguments 47
 - length in CLI applications 47
- SUBSTRING scalar function 174
- SysSchema CLI/ODBC configuration keyword 372
- system catalogs
 - querying 167

T

- TableType CLI/ODBC configuration keyword 374
- TAN scalar function
 - list 174
- targets
 - logical nodes 299
- TempDir CLI/ODBC configuration keyword 374

- termination
 - CLI application 53
 - task 17
- terms and conditions
 - use of publications 423
- threads
 - multithreaded, in CLI 133
- TIME SQL data type
 - conversion to C 399
- TIMESTAMP data type
 - conversion to C 399
- TIMESTAMPADD scalar function 174
- TIMESTAMPDIFF scalar function
 - description 174
- Trace CLI/ODBC configuration keyword 375
- TraceComm CLI/ODBC configuration keyword 376
- TraceErrImmediate CLI/ODBC configuration keyword 377
- TraceFileName CLI/ODBC configuration keyword 378
- TraceFlush CLI/ODBC configuration keyword 379
- TraceFlushOnError CLI/ODBC configuration keyword 379
- TraceLocks CLI/ODBC configuration keyword 380
- TracePathName CLI/ODBC configuration keyword 381
- TracePIDList CLI/ODBC configuration keyword 382
- TracePIDTID CLI/ODBC configuration keyword 383
- TraceRefreshInterval CLI/ODBC configuration keyword 383
- traces
 - CLI/ODBC/JDBC 189
- TraceStmtOnly CLI/ODBC configuration keyword 384
- TraceTime CLI/ODBC configuration keyword 385
- TraceTimestamp CLI/ODBC configuration keyword 386
- transaction managers
 - CLI applications
 - configuration 138
 - programming considerations 141
- transactions
 - commit or rollback 30
 - ending in CLI 32
- troubleshooting
 - online information 423
 - tutorials 423
- TRUNCATE or TRUNC scalar function
 - overview 174
- truncation
 - output strings 47
- Trusted_Connection CLI/ODBC configuration keyword 386
- tutorials
 - troubleshooting and problem determination 423
 - Visual Explain 422
- two-phase commit
 - CLI 137
- TxnIsolation CLI/ODBC configuration keyword 388

U

- UCASE scalar function 174
- UDTs (user-defined types)
 - description 147
 - in CLI 148
- UID CLI/ODBC configuration keyword 388
- Underscore CLI/ODBC configuration keyword 389
- Unicode (UCS-2)
 - CLI
 - applications 143
 - functions 144
 - ODBC driver managers 145
- uninstalling
 - IBM DB2 Driver for ODBC and CLI 273

- units of work (UOW)
 - distributed 30
- UNIX
 - setting up ODBC environment 210
- unixODBC driver manager
 - build scripts 213
 - configurations 213
 - setting up 212
- updates
 - bulk data, with bookmarks in CLI 107
 - data in CLI 35
 - DB2 Information Center 421
 - Information Center 421
- UseOldStpCall CLI/ODBC configuration keyword 390
- USER scalar function 174
- user-defined types (UDTs)
 - in CLI 148
- UseServerMsgSP CLI/ODBC configuration keyword 391

V

- VARBINARY SQL data type
 - conversion to C 399
- VARCHAR data type
 - conversion to C 399
- VARGRAPHIC data type
 - conversion to C 399
- vendor escape clauses 171
- Visual Explain
 - tutorial 422

W

- WarningList CLI/ODBC configuration keyword 391
- WEEK scalar function
 - listed 174
- WEEK_ISO scalar function
 - listed 174
- Windows
 - CLI environment 215
 - setting up CLI environment 215

X

- X/Open CAE 51
- X/Open Company 3
- X/Open SQL CLI 3
- XA
 - IBM DB2 Driver for ODBC and CLI 265
 - registering XA library with DTC 252
- XML
 - declarations
 - CLI applications 117
- XML data
 - CLI applications 117
 - inserting 118
 - retrieval 119
 - updating 118
- XML data retrieval
 - CLI applications 119
- XML data type
 - CLI applications 117
- XML declarations
 - CLI applications 117
- XML parsing
 - CLI applications 118

- XML serialization
 - CLI applications 117, 119
- XMLDeclaration CLI/ODBC configuration keyword 392

Y

- YEAR scalar function
 - list 174

Contacting IBM

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts at <http://www.ibm.com/planetwide>

To learn more about DB2 products, go to <http://www.ibm.com/software/data/db2/>.



Printed in USA

SC10-4224-00



DB2 V9 BETA

Spine information:

DB2 V9.9.0.ETA

IBM DB2

DB2 Version 9

CLI Guide and Reference, Volume 1

